Joanna Zheng

CISC 3225 Data Tools and Algorithms

Professor Matthew McNeil

INTRODUCTION

I decided to do my project on the Pokémon dataset, retrieved from Kaggle (https://www.kaggle.com/datasets/mihirbindal/the-complete-pokemon-dataset). The data was scrapped by the creator from Pokemondb, Bulbapedia, and Serebii, which are websites that contains various information about each Pokémon in the Pokédex. It contains Pokédex numbers, which are akin to IDs, unique Pokémon names, which generation is the Pokémon from, Pokémon classification groups, possible abilities, height in meters, weight in kilograms, primary and secondary types, base total of stats, battle statistics, defense multiplier against each type, capture rate, base egg steps, base happiness, and if the Pokémon is legendary, mythical, or mega. There is a lot of information to keep track of and not all will be used in this analysis.

Personally, I used to be an avid Pokémon fan. I wanted to find out how the distribution of different categories and types of Pokémon varies based off different factors, such as game generations, primary and secondary types, and classification. I am particularly interested in Pokémon battle statistics in each type and classification and analysis around capture rates. Additionally, I want to see if there are any relationships and correlations based on those data. Finally, I hope to train a model to predict a given Pokémon's primary type using its battle statistics and physiques. My goal is to determine whether these non-legendary and mythical Pokémon seem more manually crafted or are randomly distributed and generated and if there are any patterns to how certain battle stats and types are given.

Upon scrolling through the data, I realized that it needs a lot of cleaning. There is a lot of data that does not help with my analysis. I dropped a lot of columns, particularly the defense multipliers against each type, base total steps, base happiness, is_legendary, is_mythical, is_mega, and abilities. This is because it will result in a mass number of redundant columns. I deliberately decided not to research legendary and mythical Pokémon because their stats tend to be better than normal ones due to their lore relevancy and rarity. They are also very region and generation specific and often only 1 classification would categorize those Pokémon, resulting in more redundant values which I will get to further down the research. Mega Pokémon are also stronger and often are evolved from megastones and the change is not permanent. As there are catalysts for more stats for the same Pokémon on the pokedex, it may appear as data may have some sort of bias, thus it was removed. This data also had several NaN

values in which were deleted except for rows with None in type2. Filling in the null values for numeric values will not yield accurate results. Column 'capture_rate' had empty values that must be replaced and deleted as well, otherwise it will not convert to numeric values. Our remaining data is as follows:

```
pkmn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 871 entries, 0 to 871
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   index           871 non-null    Int64
 1   pokedex_number  871 non-null    Int64
 2   name            871 non-null    string
 3   generation      871 non-null    Int64
 4   classfication   871 non-null    string
 5   height_m        871 non-null    Float64
 6   weight_kg       871 non-null    Float64
 7   type1           871 non-null    string
 8   type2           437 non-null    string
 9   base_total      871 non-null    Int64
 10  hp              871 non-null    Int64
 11  attack          871 non-null    Int64
 12  defense         871 non-null    Int64
 13  sp_attack       871 non-null    Int64
 14  sp_defense      871 non-null    Int64
 15  speed           871 non-null    Int64
 16  capture_rate    871 non-null    Int64
dtypes: Float64(2), Int64(11), string(4)
memory usage: 133.5 KB
```

Now that we are done with cleaning the data, it is time to explore.

We will first explore the mean, median, minimum, maximum, quartile values, and the standard deviation. Generally, we can ignore the python given index, pokedex number, and generation. Although those are numeric, they are not important in this context. Before we dive in, I want to talk a little bit about Pokémon biology. Pokémon in their world is equivalent to the animal species in our world. There are different sizes and shapes, different compositions and physiques, and interestingly, they can be made of different types of matter too. What I found interesting is that the minimum height of a Pokémon is 0.1 meters, which is about 4 inches, and that is really small. Yet a small Pokémon can take part in the battlefield? I queried for the list of Pokémon that are that tiny, and indeed there are several potent Pokémon that are that small, most of which are Bug type like Joltik, and supernatural types like Fairy (Flabebe) and Ghost

(Sinistea). Each Pokémon is given a set of moves which they can use to survive in the wild and be used on the battlefield. I'm genuinely surprised that these Pokémon pack a punch for being so tiny! The tallest Pokémon is 14.5 meters, which is about 47.6 feet. That's about as tall as a 4-5 story building. What's even more surprising is that it's not even the heaviest Pokémon either. So, what can be the relationship between height and weight? We will get to this when we talk about the correlations. Moving onwards, any values towards the minimum for battle stats—hp, attack, defense, and etc—are usually unevolved Pokémon since they start off low. Although that is the case, the minimum for hp is 1.0 hit point. Keep in mind these are base stats, which means it's the stats they will have when they first hatch from an egg, or for evolved Pokémon, when they first evolve at the lowest level possible. This level 0 Shedinja has 1 hp, which I did not even think was possible. If we consider Pokémon ability, Shedinja's low hp is nothing when it has Wonder Guard, which means only types effective against its Bug and Ghost dual type can hit it. With this other information, it does make sense why it has low stats. Below are the rest of the data:

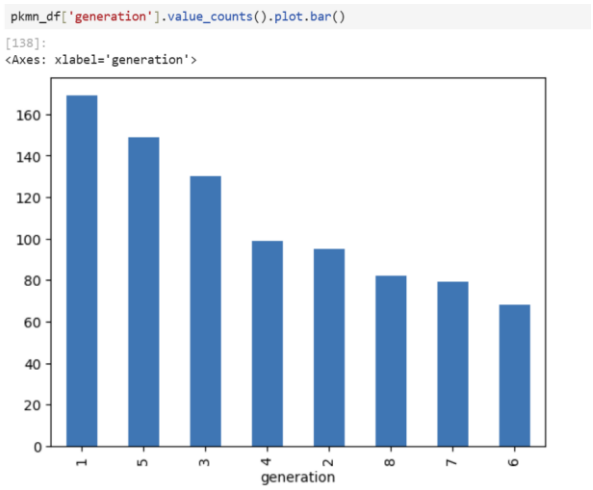
```
[132]: pkmn_df.describe()
```

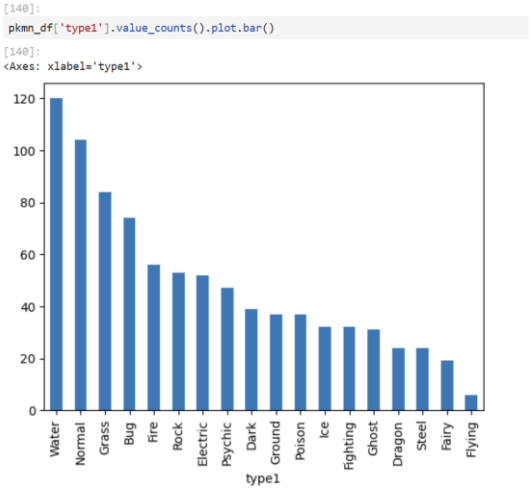
[132]:

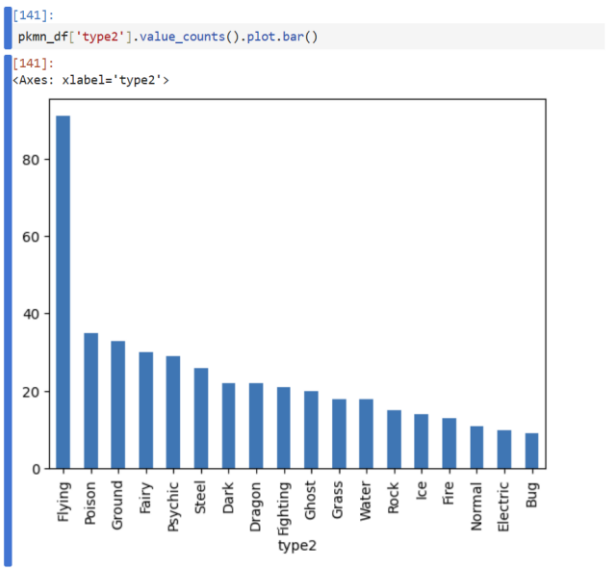
	index	pokedex_number	generation	height_m	weight_kg	base_total	hp	attack	defense	sp_attack	sp_defense	speed	capture_rate
count	871.0	871.0	871.0	871.0	871.0	871.0	871.0	871.0	871.0	871.0	871.0	871.0	871.0
mean	499.529277	430.962113	4.026406	1.093685	51.58806	408.597015	66.576349	74.475316	69.869116	66.495982	67.270953	63.9093	105.344432
std	295.552965	261.3945	2.267814	1.060652	92.043385	99.681167	24.8605	28.92756	28.557807	27.432545	25.050479	27.241192	74.272569
min	0.0	1.0	1.0	0.1	0.1	175.0	1.0	5.0	5.0	10.0	20.0	5.0	3.0
25%	237.5	200.5	2.0	0.5	8.15	318.0	50.0	54.5	50.0	45.0	50.0	43.0	45.0
50%	501.0	428.0	4.0	0.9	24.5	420.0	65.0	70.0	65.0	61.0	65.0	60.0	75.0
75%	760.5	660.5	6.0	1.4	55.9	490.0	78.0	94.5	85.0	85.0	81.0	84.0	180.0
max	1007.0	887.0	8.0	14.5	999.9	700.0	255.0	181.0	230.0	173.0	230.0	160.0	255.0

EXPLORATORY ANALYSIS

We will move on to preliminary data visualization. This will be based on all the categorical variables besides the unique Pokémon name and the classifications, which have large numbers of values.







From our bar graphs based on Generation, we can see that the creation of new Pokémon has decreased when generation count goes up. Each generation and its related region often have older Pokémon that are reused to fill in the role gaps. Think of newer Pokémon as native species to that region and older ones being introduced by outside factors or common species that show up globally. When we look at Type 1, we see that Water, Normal, Grass, and Bug are more heavily skewed towards the primary type. The primary type, in most cases, is used to determine

characteristics and composition of a Pokémon. When we look at Type 2, Flying type dominates all the other types. Looking back at Type 1, Flying type is the least of primary types.

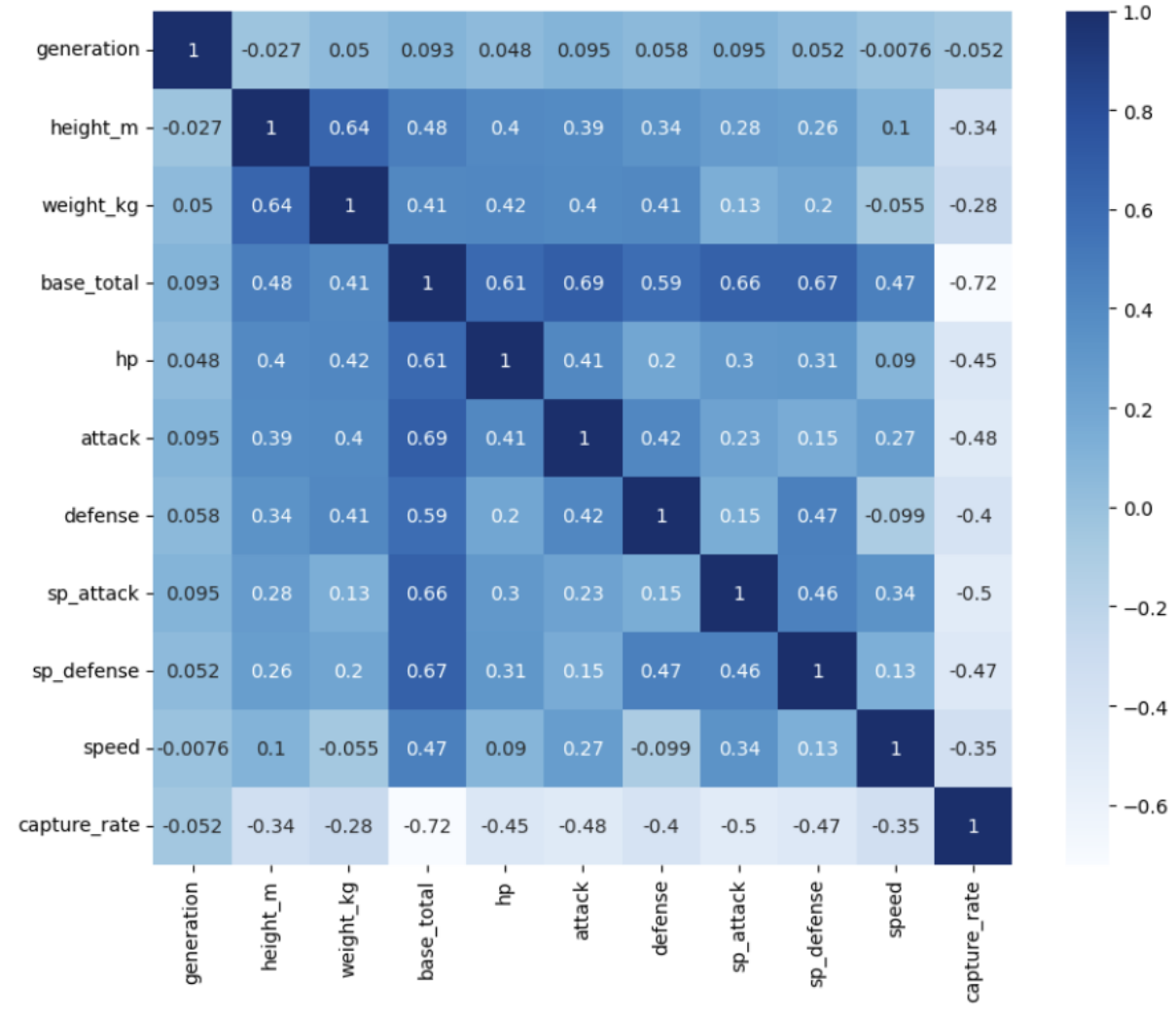
Next, the data is preprocessed and normalized. Using Matplotlib and Seaborn, a heatmap is made into a correlation chart. Let's talk a little about the data correlations.

[148]:

```
plt.figure(figsize=(10,8))
sns.heatmap(pk_numeric.corr(), annot=True, cmap="Blues")
#Couldn't do on the categoricals, would result in extremely large matrices
#However, my hypothesis is that height and weight may have dependence on classific
```

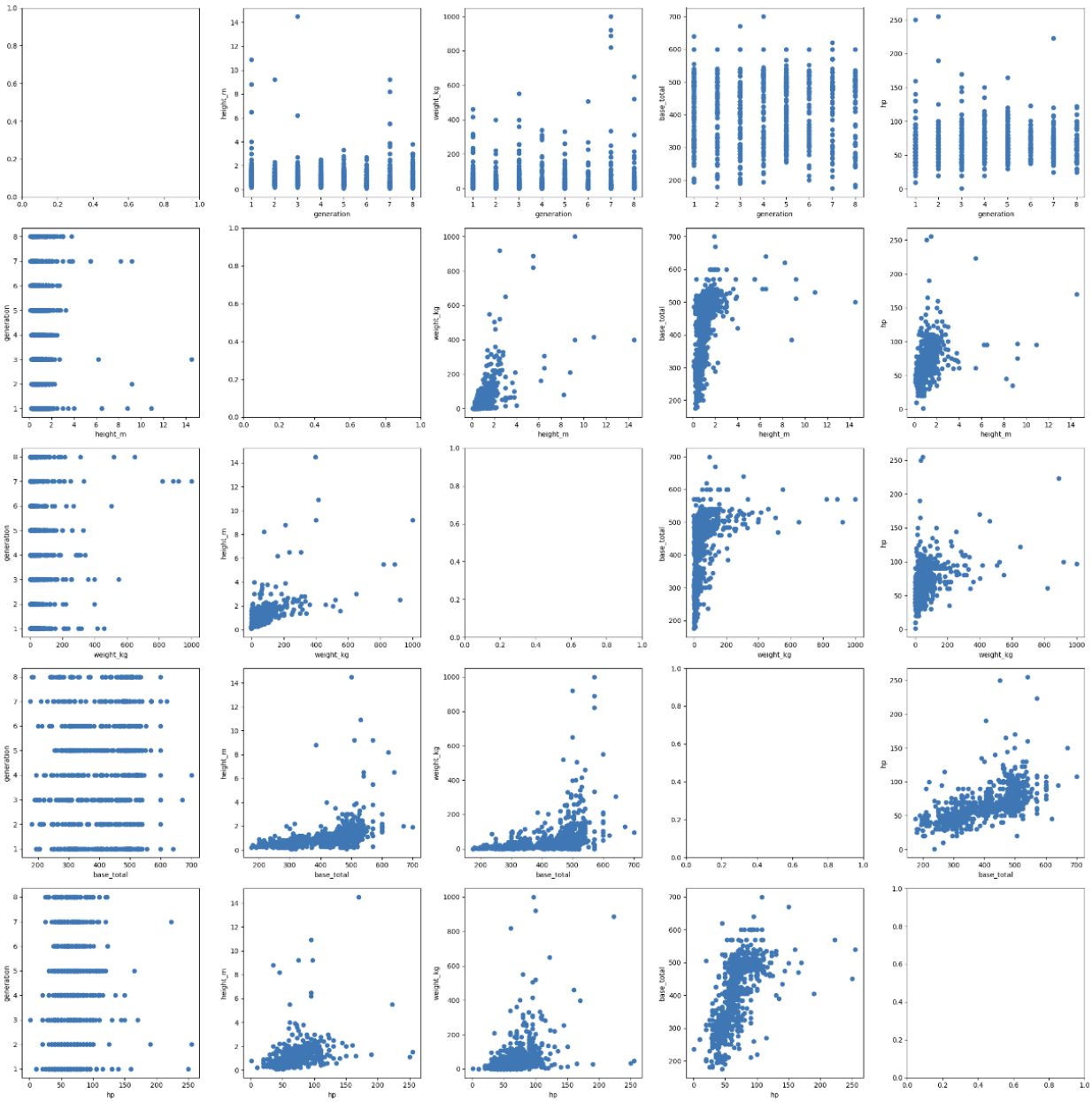
[148]:

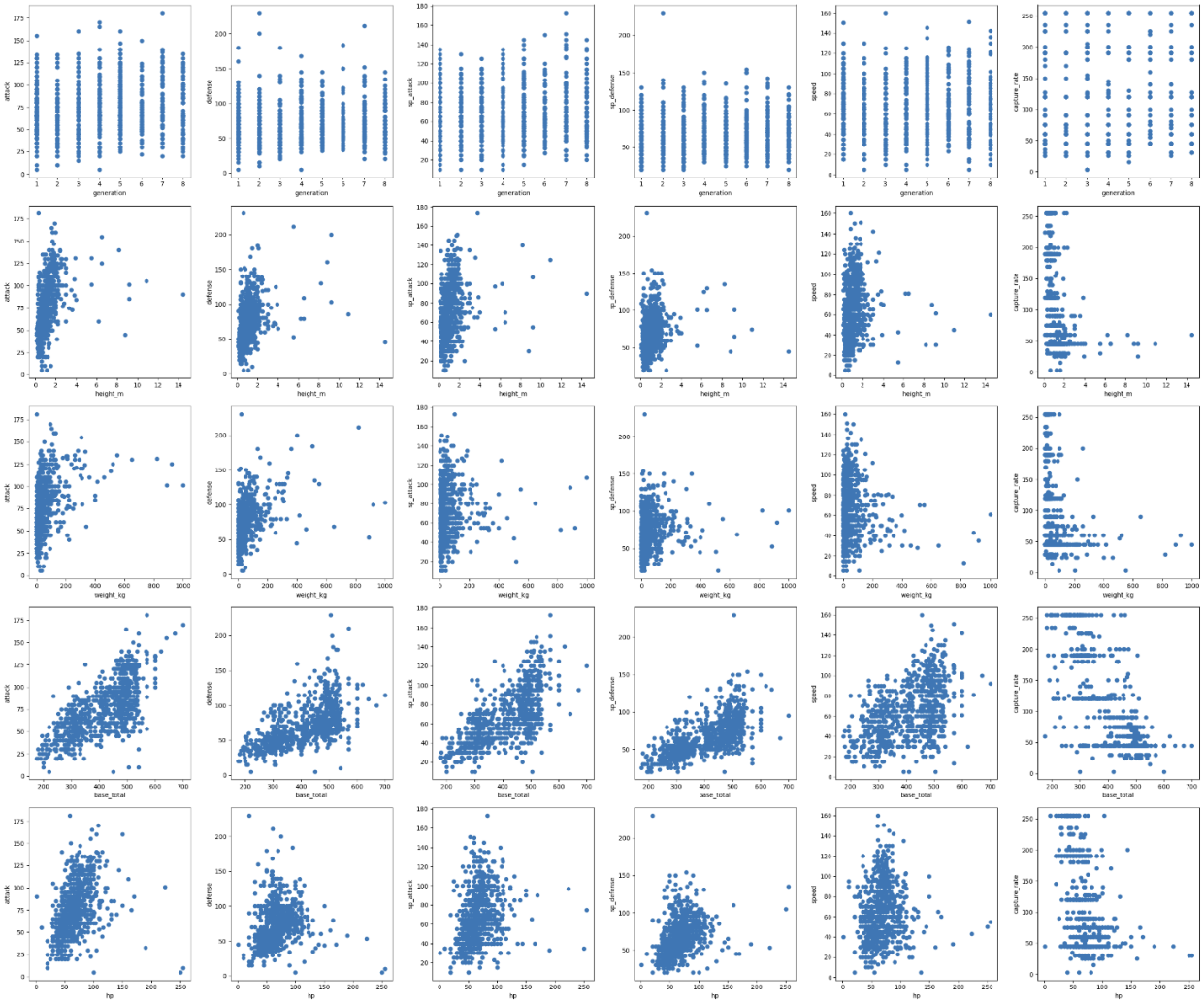
<Axes: >

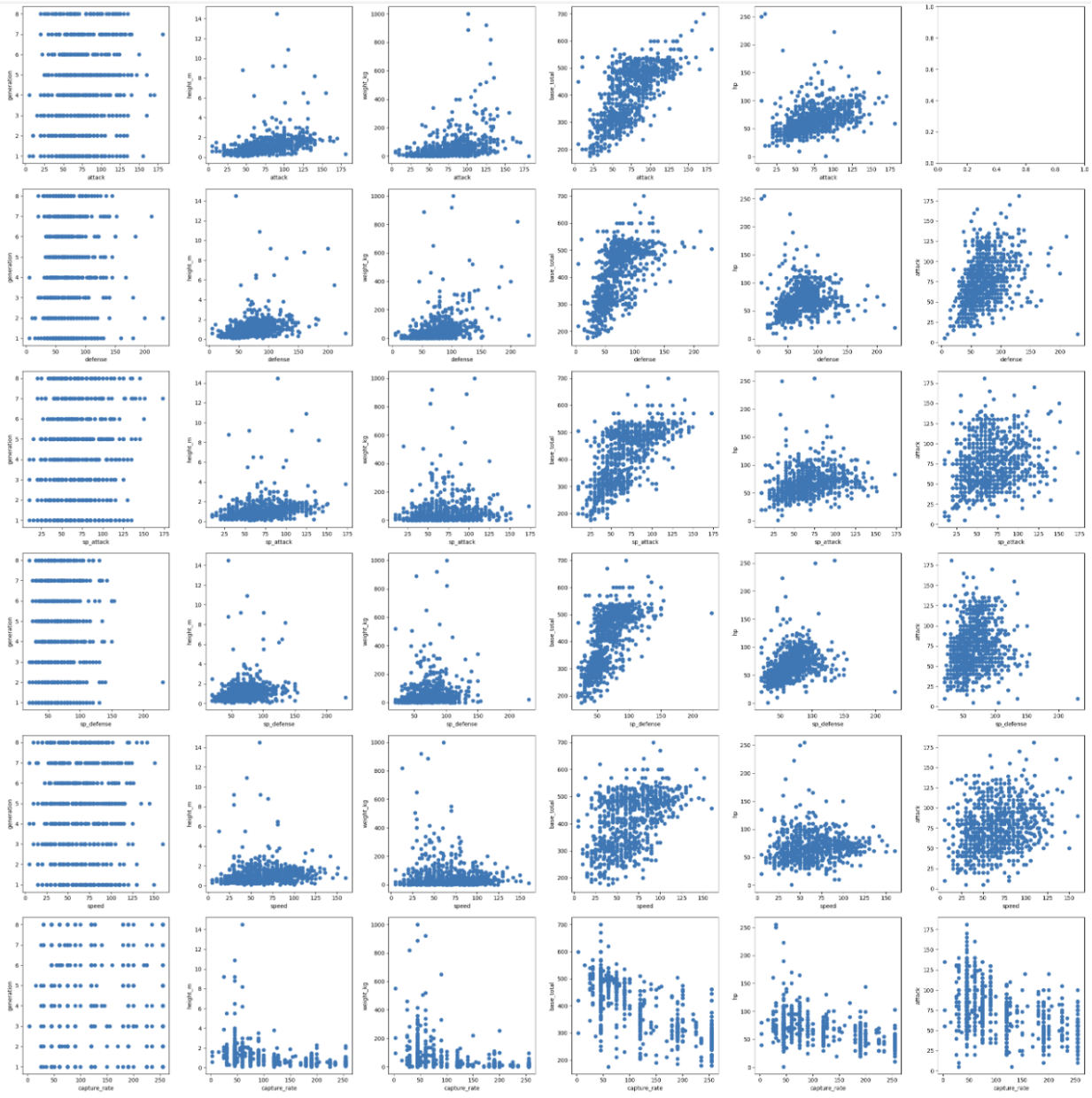


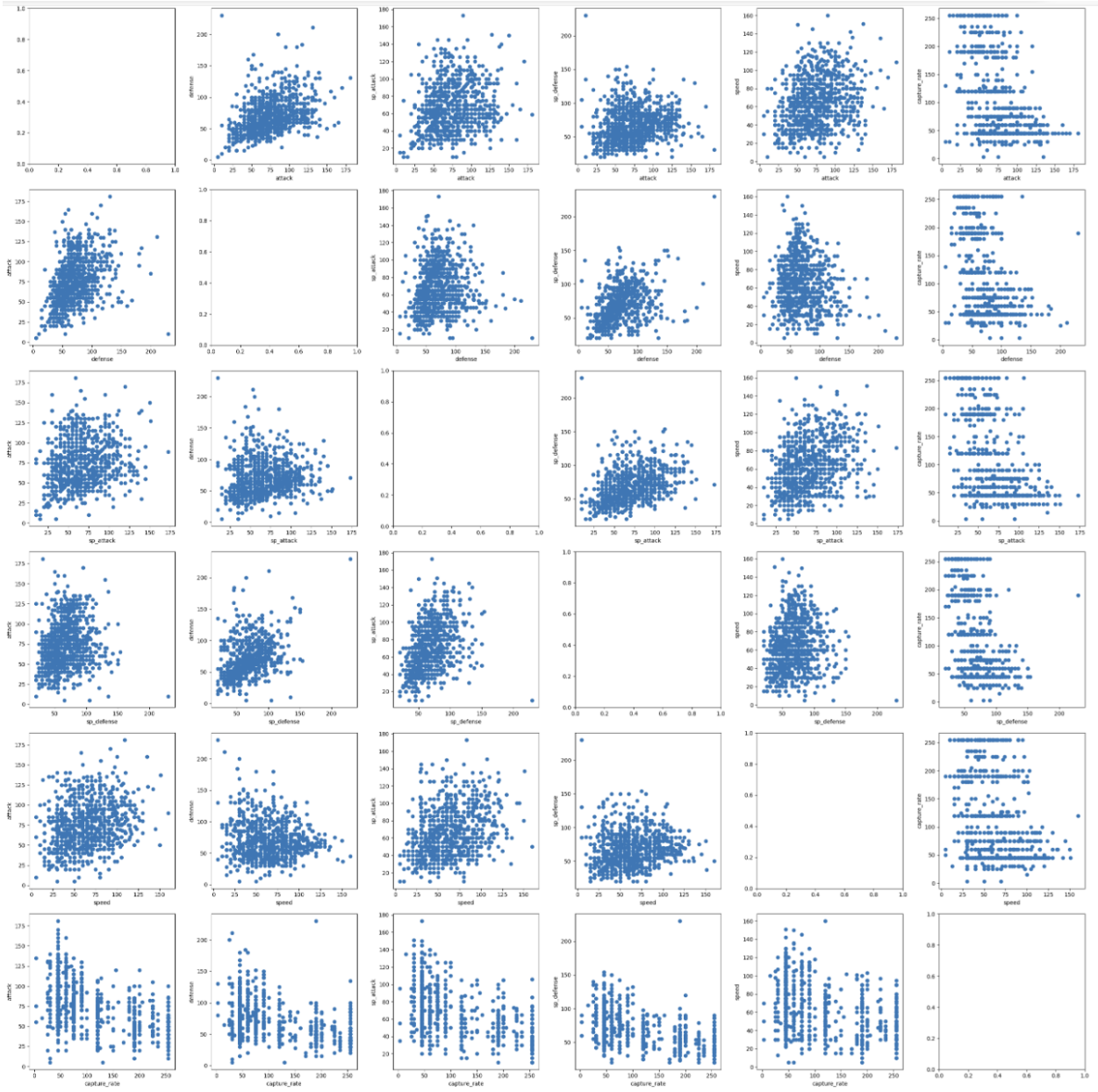
Categorical values are left out of the correlation matrices due to the nature of one-hot encoding. It will result in many squares and take a really long time to process and load because of the column size. Before looking at the matrix, I made some predictions. I predicted that the height and weight would be positively correlated. Indeed, the height and weight correlation have a value of positive 0.64, which is high enough to say it is likely that as height grows larger, so will the weight and vice versa. It is a valid claim for a large sample of entities, although it is not entirely true if we go back to our example query of Whailord being the tallest Pokémon, yet isn't the weightiest one. I also made a prediction that higher base total is positively correlated with all the battle stats, and negatively correlated with capture rate. The reasoning is that a higher base total means a Pokémon has more points for any of the stats. The stronger the Pokémon, the less chance of it being captured. We talked about how Pokémon have different evolutions and generally, the higher the stage of evolution, the more stats it has, and the least likely it can be captured by a normal Pokéball. The only stat being under 0.50 is speed but is still big enough to be considered. Other correlations that can be found here are between special attack and special defense, attack and defense, and hp and attack. Generation seems to have nothing to do with any of these stats, so it can be safe to say that Pokémon are not stronger or weaker and better in terms of stats as the generation count increases. This is good because there is no power creep, which is a term used in games when an entity from early on in game is weaker and less effective than one from late game. An interesting find in this correlation matrix that completely debunked the idea that weight is strongly negatively correlated with speed, but it is not. In fact, it shows weak negative correlation, so we can have a really heavy Pokémon that moves at a high speed. For example, if we take the average weight plus the standard deviation with speed = 110, we get two Pokémon that weigh a lot but move fast: Scolipede and Naganadel. Similarly, it can be said for Pokémon that move slower but weigh under 5 kilogram, like Litwick, Cleffa, and Solosis.

Next, we will move on to some scatter plots to further visualize our findings using the correlation matrix.


generation
height_m
weight_kg
base_total
hp

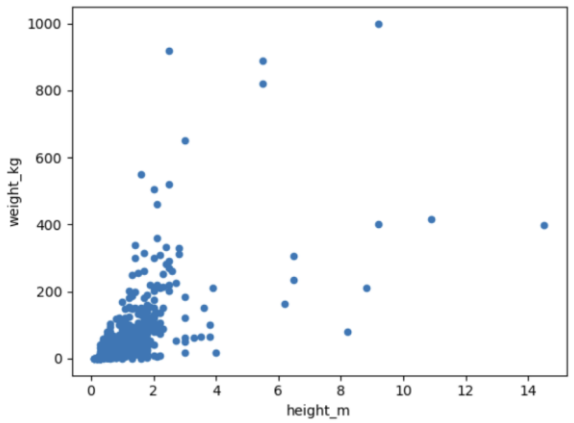

attack
defense
sp_attack
sp_defense
speed
capture_rate
generation
height_m
weight_kg
base_total
hp


generation
height_m
weight_kg
base_total
hp
attack
defense
sp_attack
sp_defense
speed
capture_rate

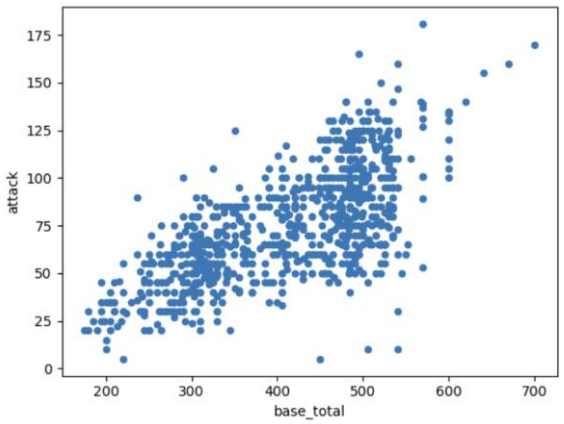


If we zoom in on a couple of of them:

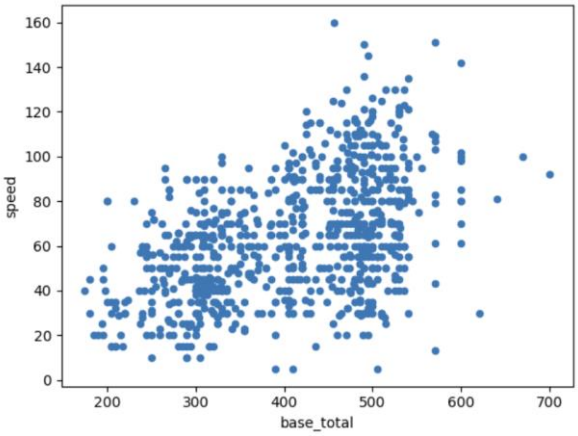
<Axes: xlabel='height_m', ylabel='weight_kg'>



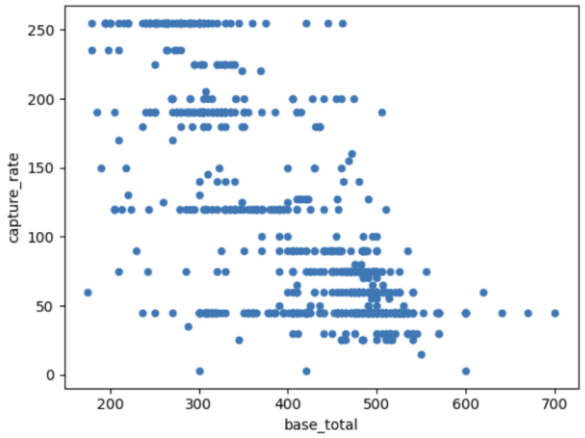
<Axes: xlabel='base_total', ylabel='attack'>



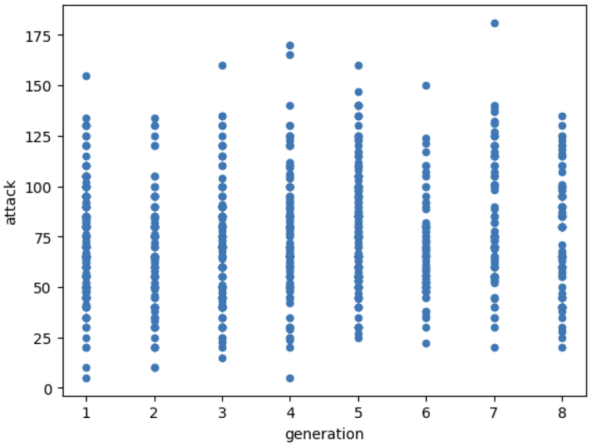
<Axes: xlabel='base_total', ylabel='speed'>



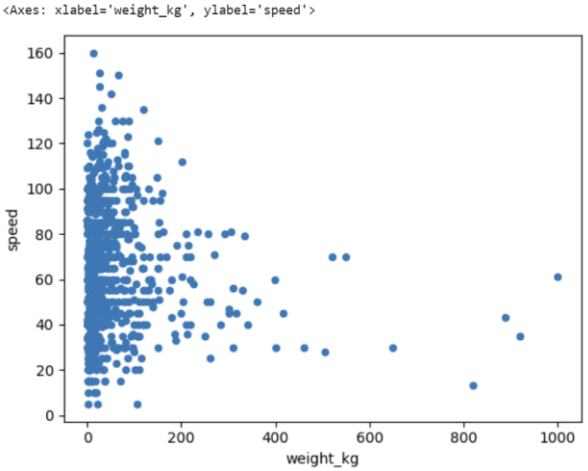
<Axes: xlabel='base_total', ylabel='capture_rate'>



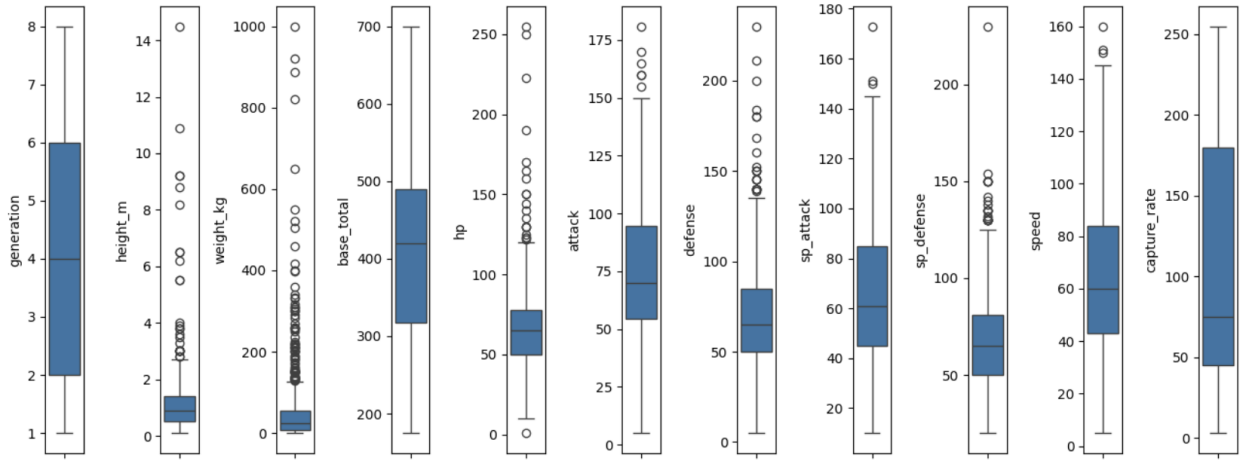
<Axes: xlabel='generation', ylabel='attack'>



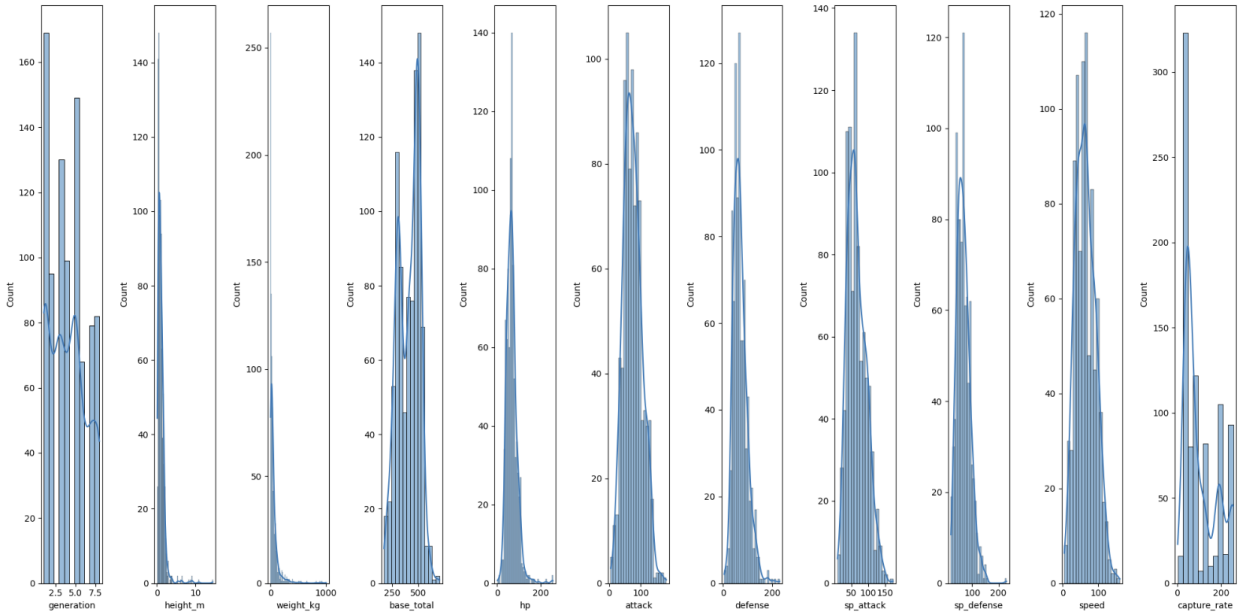
<Axes: xlabel='weight_kg', ylabel='speed'>



You can see that ones that have stronger correlation tend to have more defined centerline compared to those that have weak correlations, which then are typically more scattered and less defined.



Looking at the box and whisker plots, we see that there are several high values dotted on the visuals. Especially when you look at height and weight, there seems to be a lot of values that are out of the range of the left and right standard deviations. These are still important data despite not being within standard deviation, so they shouldn't be removed.



The histogram further shows that a lot of these distributions are not normal distributions, especially the height, weight, hp, sp_defense, generation (which is irrelevant in this analysis), and capture_rate. Base_total, attack, defense, sp_attack, and speed appears to be the more close-to-normal distributions. Not having a normal distribution signifies that there are a lot of values that fall outside of the standard deviation from the mean, which can be seen in the box and whisker plots as well.

DEEPER ANALYSIS

1. Answering questions with Pandas features querying

There are many lingering questions. A bit of querying will answer those questions.

- Question 1: How many Pokémon are in each classification?

```
pkmn_class_ct = pkmn_df.groupby('classfication')['pokedex_number'].count().reset_index(name='count').sort_values(by='count', ascending=False)
pkmn_class_ct
```

[307]:

	classfication	count
336	Mouse Pokemon	11
204	Fox Pokemon	9
151	Dragon Pokemon	8
373	Poison Pin Pokemon	6
364	Plasma Pokemon	6
340	Mushroom Pokemon	6
27	Balloon Pokemon	5
203	Fossil Pokemon	5
332	Mole Pokemon	5

There are 581 classifications, and there are 871 Pokémon in this dataset total. That is about 66.70% ratio. Furthermore, when the count is queried for classifications that only have 1 Pokémon in it, there is 395 of them. This will be taken into consideration when we run future analysis.

On a following query, I asked which classifications have greater than 5. It is not a lot.

```
pkmn_class_ct[pkmn_class_ct['count'] >= 5]
```

	classfication	count
336	Mouse Pokemon	11
204	Fox Pokemon	9
151	Dragon Pokemon	8
373	Poison Pin Pokemon	6
364	Plasma Pokemon	6
340	Mushroom Pokemon	6
27	Balloon Pokemon	5
203	Fossil Pokemon	5
332	Mole Pokemon	5

- Question 2: What are the average height and weight and battle stats of each Pokémon classification with a count of 5 and up?

```
cls_ct_filter = pd.merge(fiveup, pkmn_df, how='left', on='classfication')
cls_ct_filter.groupby('classfication')[['height_m', 'weight_kg', 'hp', 'attack', 'defense', 'sp_attack', 'sp_defense', 'speed']].mean()
```

classfication	height_m	weight_kg	hp	attack	defense	sp_attack	sp_defense	speed
Balloon Pokemon	0.54	4.72	100.0	58.0	39.8	57.0	38.8	47.0
Dragon Pokemon	1.8625	83.4875	67.25	89.625	76.875	87.5	76.875	76.25
Fossil Pokemon	2.04	157.8	88.0	97.0	89.0	76.0	79.0	78.0
Fox Pokemon	0.9	16.811111	56.222222	53.777778	54.0	73.555556	78.222222	79.555556
Mole Pokemon	0.42	22.04	30.0	79.0	41.0	40.0	55.0	96.6
Mouse Pokemon	0.672727	22.254545	54.090909	76.727273	68.181818	43.181818	56.363636	76.181818
Mushroom Pokemon	0.616667	15.016667	66.333333	79.166667	65.0	57.5	65.0	34.166667
Plasma Pokemon	0.3	0.3	50.0	62.5	102.0	103.333333	102.0	86.833333
Poison Pin Pokemon	1.133333	34.55	62.0	64.0	59.333333	65.0	55.0	67.666667

- Question 3: For classifications that have exactly 6 Pokémon, which one has the highest average battle stat? Lowest?

HP: Highest – Mushroom Pokémon; Lowest – Plasma Pokémon

Attack: Highest – Mushroom Pokémon; Lowest – Plasma Pokémon

Speed: Highest – Plasma Pokémon; Lowest – Mushroom Pokémon

Height: Highest – Poison Pin Pokémon; Lowest – Plasma Pokémon

Weight: Highest – Poison Pin Pokémon; Lowest – Plasma Pokémon

Defense: Highest – Plasma Pokémon; Lowest – Poison Pin Pokémon

Special Attack: Highest – Plasma Pokémon; Lowest – Mushroom Pokémon

Special Defense: Highest – Plasma Pokémon; Lowest – Poison Pin Pokémon

```
filtered = pkmn_df.groupby('classfication').filter(lambda x: len(x) == 6)
filtered.groupby('classfication')['hp'].mean() #Mishroom Pokemon, PLasma Pokemon
filtered.groupby('classfication')['attack'].mean() #Mushroom Pokemon, Plasma Pokemon
filtered.groupby('classfication')['speed'].mean() #Plasma Pokemon, Mushroom Pokemon
filtered.groupby('classfication')['height_m'].mean() #Poison Pin Pokemon, Plasma Pokemon
filtered.groupby('classfication')['weight_kg'].mean() #Poison Pin Pokemon, Plasma Pokemon
filtered.groupby('classfication')['defense'].mean() #Plasma Pokemon, Poison Pin Pokemon
filtered.groupby('classfication')['sp_attack'].mean() #Plasma Pokemon, Mushroom Pokemon
filtered.groupby('classfication')['sp_defense'].mean() #Plasma Pokemon, Poison Pin Pokemon
```

- Question 4: What are the average height and weight of each Pokémon primary type? Hp, attack, defense, special defense, special attack, and speed?

```
prim_type_stats = pkmn_df.groupby('type1')[['height_m', 'weight_kg', 'hp', 'attack', 'defense', 'sp_attack', 'sp_defense', 'speed']].mean()
prim_type_stats
```

type1	height_m	weight_kg	hp	attack	defense	sp_attack	sp_defense	speed
Bug	0.855405	31.997297	56.054054	65.797297	69.27027	55.594595	62.175676	61.527027
Dark	1.020513	51.810256	67.74359	77.384615	61.512821	63.948718	63.384615	73.307692
Dragon	1.441667	60.804167	68.416667	96.375	73.25	68.25	72.0	73.666667
Electric	0.846154	32.138462	59.192308	70.711538	64.653846	84.038462	69.961538	80.365385
Fairy	0.621053	11.042105	68.157895	56.157895	64.578947	76.105263	86.0	48.684211
Fighting	1.234375	58.278125	70.53125	97.59375	68.0	50.5	63.6875	61.625
Fire	1.057143	45.728571	66.517857	80.214286	63.696429	79.785714	66.767857	72.982143
Flying	1.0	33.966667	66.5	64.333333	60.833333	59.333333	65.0	77.333333
Ghost	1.0	35.703226	59.225806	68.193548	75.709677	82.967742	78.774194	56.709677
Grass	1.02381	35.439286	65.142857	71.940476	69.892857	71.845238	67.452381	56.833333
Ground	1.2	91.435135	68.297297	88.081081	81.72973	47.783784	62.081081	58.432432
Ice	1.125	85.4375	70.4375	75.4375	69.8125	66.9375	66.59375	66.625
Normal	0.932692	36.636538	75.192308	72.153846	57.480769	53.759615	60.615385	66.730769
Poison	1.237838	36.127027	67.27027	74.216216	70.972973	63.783784	66.405405	64.405405
Psychic	0.906383	24.755319	65.255319	52.723404	58.468085	84.851064	76.340426	64.893617
Rock	1.326415	113.190566	66.603774	87.811321	95.339623	58.584906	69.113208	53.981132
Steel	1.783333	174.316667	64.416667	88.291667	108.916667	63.75	71.0	48.75
Water	1.320833	49.009167	68.841667	71.6	69.841667	69.375	67.508333	63.566667

- Question 5: Which primary type has the highest mean of each stat? Lowest? Display the values as well.

Height_m: Steel 1.78

Weight_kg: Steel 174.32

HP: Normal 75.19

Attack: Fighting 97.59

Defense: Steel 108.92

Sp_attack: Psychic 84.85

Sp_defense: Fairy 86.0

Speed: Electric 80.37

- Question 6: Which primary type has the lowest mean of each stat? Lowest? Display the values as well.

Height_m: Fairy 0.62

Weight_kg: Fairy 11.04

HP: Bug 56.05

Attack: Psychic 52.72

Defense: Normal 57.48

Sp_attack: Ground 47.78

Sp_defense: Normal 60.62

Speed: Fairy 48.68

- Question 7: #Which primary type is most common in each generation?

```
pkmn_df.groupby(['generation', 'type1'])['pokedex_number'].count().reset_index(name='count').sort_values(by=['count','generation'], ascending=False)
#Gen 1: Water 39 | Gen 2: Water 17 | Gen 3: Water 24 | Gen 4: Normal 15
#Gen 5: Bug 17 & Water 17 | Gen 7: Grass 11 | Gen 8: Electric 10
```

Gen 1: Water 39 | Gen 2: Water 17 | Gen 3: Water 24 | Gen 4: Normal 15

Gen 5: Bug 17 & Water 17 | Gen 7: Grass 11 | Gen 8: Electric 10

- Question 8: What is the most common secondary type that is not null? Least Common?

Most common: Flying 91

Least common: Bug 9

- Question 9: What is a common secondary type for Grass primary types?

Poison Type

```
grass_pkmn = pkmn_df[pkmn_df['type1'] == 'Grass']
grass_pkmn['type2'].value_counts().idxmax()
```

```
'Poison'
```

2. Chi-square Analysis for the categorical values

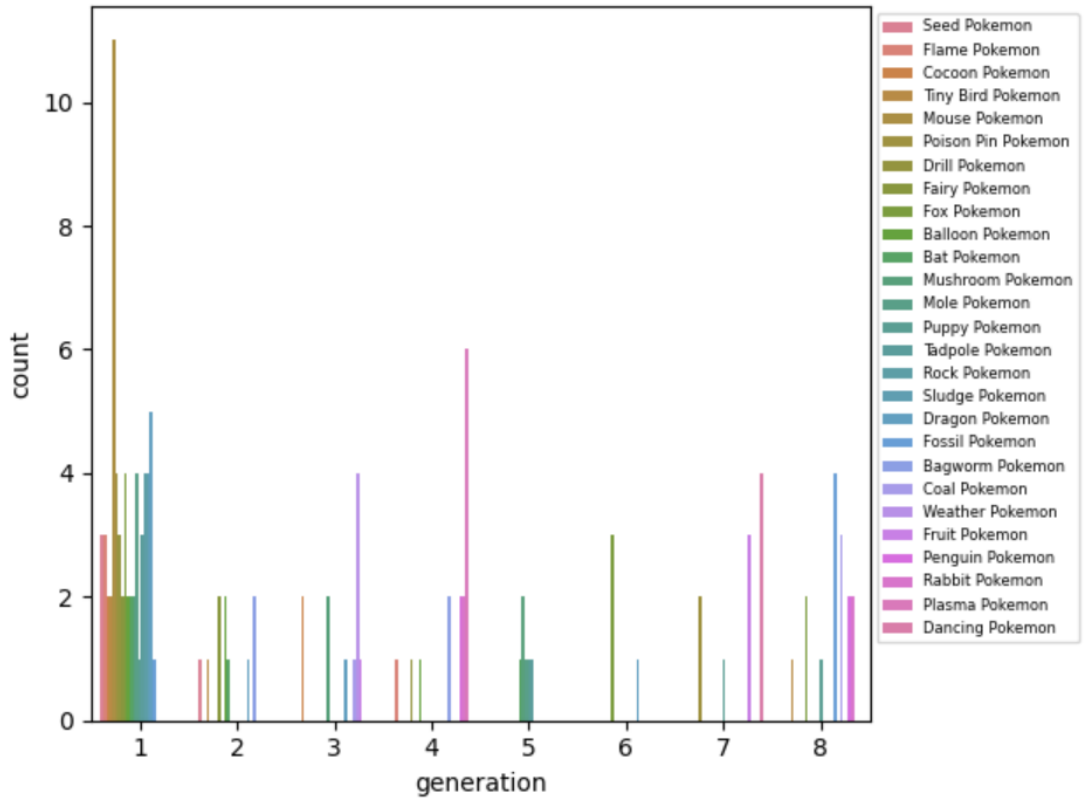
One of the method I used was chi-square statistical analysis where two hypothesis is made, H0 the null hypothesis and H1 the alternative, and the chi-square matrices will show the results. This method was specifically used on categorical values. In this test, I used three tests for different value pairings: generation and classification, generation and all types, type1 and type2.

First, I wanted to find out the relationship between generation and classification with the following hypothesis:

H0: Each generation has a near even amount of each Pokémon classification.

H1: Pokémon classifications are not evenly distributed amongst each generation.

There are 8 generations, so I conducted the analysis on only Pokémon classifications that have a count of 4 or higher. Anything less than 4 would seem unreasonable to run the test on because it would be on less than 50% of the population in the category. Generally, the higher the number of each classification, the easier it would be to prove the hypothesis because the pattern can be very apparent. A bar graph was plotted for each count:



As expected from our exploratory analysis, generation 1 has the most Pokémon to be classified. From this plot alone, it shows that some classifications are specifically made within certain generations, and some that are distributed but not evenly throughout others.

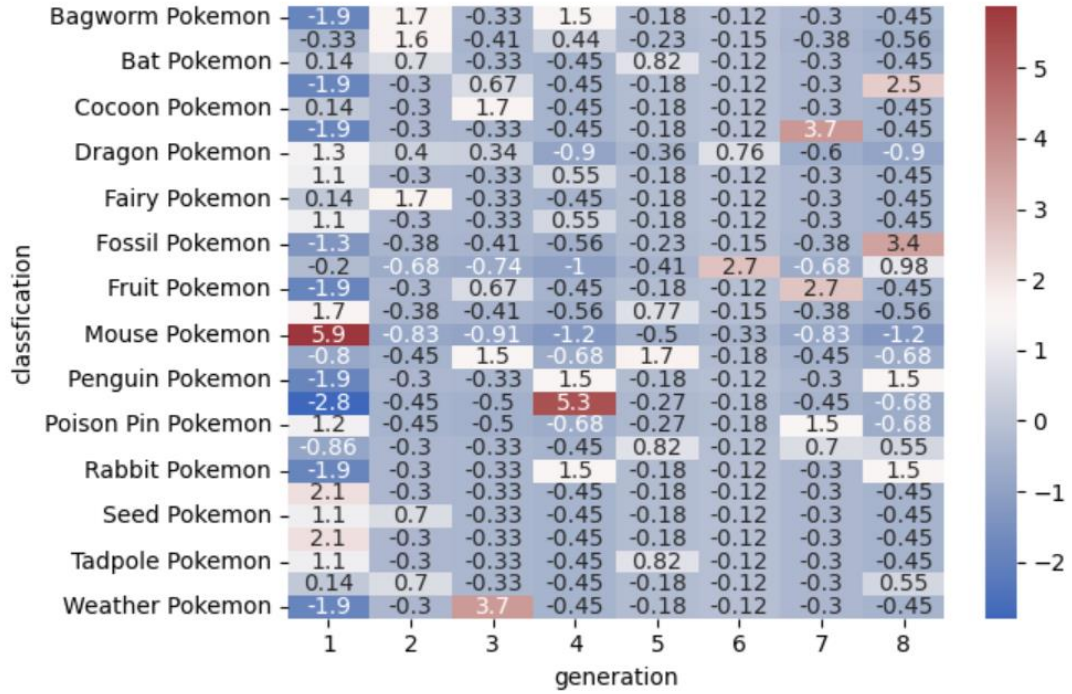
generation	1	2	3	4	5	6	7	8
classfication								
Bagworm Pokemon	0	2	0	2	0	0	0	0
Balloon Pokemon	2	2	0	1	0	0	0	0
Bat Pokemon	2	1	0	0	1	0	0	0
Coal Pokemon	0	0	1	0	0	0	0	3
Cocoon Pokemon	2	0	2	0	0	0	0	0
Dancing Pokemon	0	0	0	0	0	0	4	0
Dragon Pokemon	5	1	1	0	0	1	0	0
Drill Pokemon	3	0	0	1	0	0	0	0
Fairy Pokemon	2	2	0	0	0	0	0	0
Flame Pokemon	3	0	0	1	0	0	0	0
Fossil Pokemon	1	0	0	0	0	0	0	4
Fox Pokemon	4	0	0	0	0	3	0	2
Fruit Pokemon	0	0	1	0	0	0	3	0
Mole Pokemon	4	0	0	0	1	0	0	0
Mouse Pokemon	11	0	0	0	0	0	0	0
Mushroom Pokemon	2	0	2	0	2	0	0	0
Penguin Pokemon	0	0	0	2	0	0	0	2
Plasma Pokemon	0	0	0	6	0	0	0	0
Poison Pin Pokemon	4	0	0	0	0	0	2	0
Puppy Pokemon	1	0	0	0	1	0	1	1
Rabbit Pokemon	0	0	0	2	0	0	0	2
Rock Pokemon	4	0	0	0	0	0	0	0
Seed Pokemon	3	1	0	0	0	0	0	0
Sludge Pokemon	4	0	0	0	0	0	0	0
Tadpole Pokemon	3	0	0	0	1	0	0	0
Tiny Bird Pokemon	2	1	0	0	0	0	0	1
Weather Pokemon	0	0	4	0	0	0	0	0

The following classification of Pokémon are in three or more generations: balloon, bat, dragon, fox, mushroom, puppy, and tiny bird. This could be due to the stable introduction of new related Pokémon in each of the generations. Ultimately, regional environmental elements can determine what sort of Pokémon live there. For example, all the Rock Pokémon classifications and the majority of the Drill Pokémon were created in the Kanto region (first generation) most likely due to the vast amounts of mountain ranges. The majority of Kanto region is land filled with cities and towns, so the Mouse Pokémon is common. Also, note that Pokémon that go through evolution may stay the same classification, so it is not uncommon when 2 to 3 of the same classifications reside in the same generation or region. Upon calculating the chi2_contingency result, we see this:

```
Chi2ContingencyResult(statistic=416.8048206527642, pvalue=1.837108449348639e-20, dof=182, ex
pected_freq=array([[1.86466165, 0.30075188, 0.33082707, 0.45112782, 0.18045113,
        0.12030075, 0.30075188, 0.45112782],
       [2.33082707, 0.37593985, 0.41353383, 0.56390977, 0.22556391,
        0.15037594, 0.37593985, 0.56390977],
       [1.86466165, 0.30075188, 0.33082707, 0.45112782, 0.18045113,
        0.12030075, 0.30075188, 0.45112782],
       [1.86466165, 0.30075188, 0.33082707, 0.45112782, 0.18045113,
        0.12030075, 0.30075188, 0.45112782],
       [1.86466165, 0.30075188, 0.33082707, 0.45112782, 0.18045113,
        0.12030075, 0.30075188, 0.45112782],
       [1.86466165, 0.30075188, 0.33082707, 0.45112782, 0.18045113,
        0.12030075, 0.30075188, 0.45112782],
       [3.72932331, 0.60150376, 0.66165414, 0.90225564, 0.36090226,
        0.2406015 , 0.60150376, 0.90225564],
       [1.86466165, 0.30075188, 0.33082707, 0.45112782, 0.18045113,
        0.12030075, 0.30075188, 0.45112782],
       [1.86466165, 0.30075188, 0.33082707, 0.45112782, 0.18045113,
        0.12030075, 0.30075188, 0.45112782],
```

The p-value is significantly lower than 0.1, which means that our null hypothesis of even distribution is voided. As we see in the contingency table analysis and count plot, the distributions are not even. The matrices visual is as follows:

[211]:
<Axes: xlabel='generation', ylabel='classfication'>

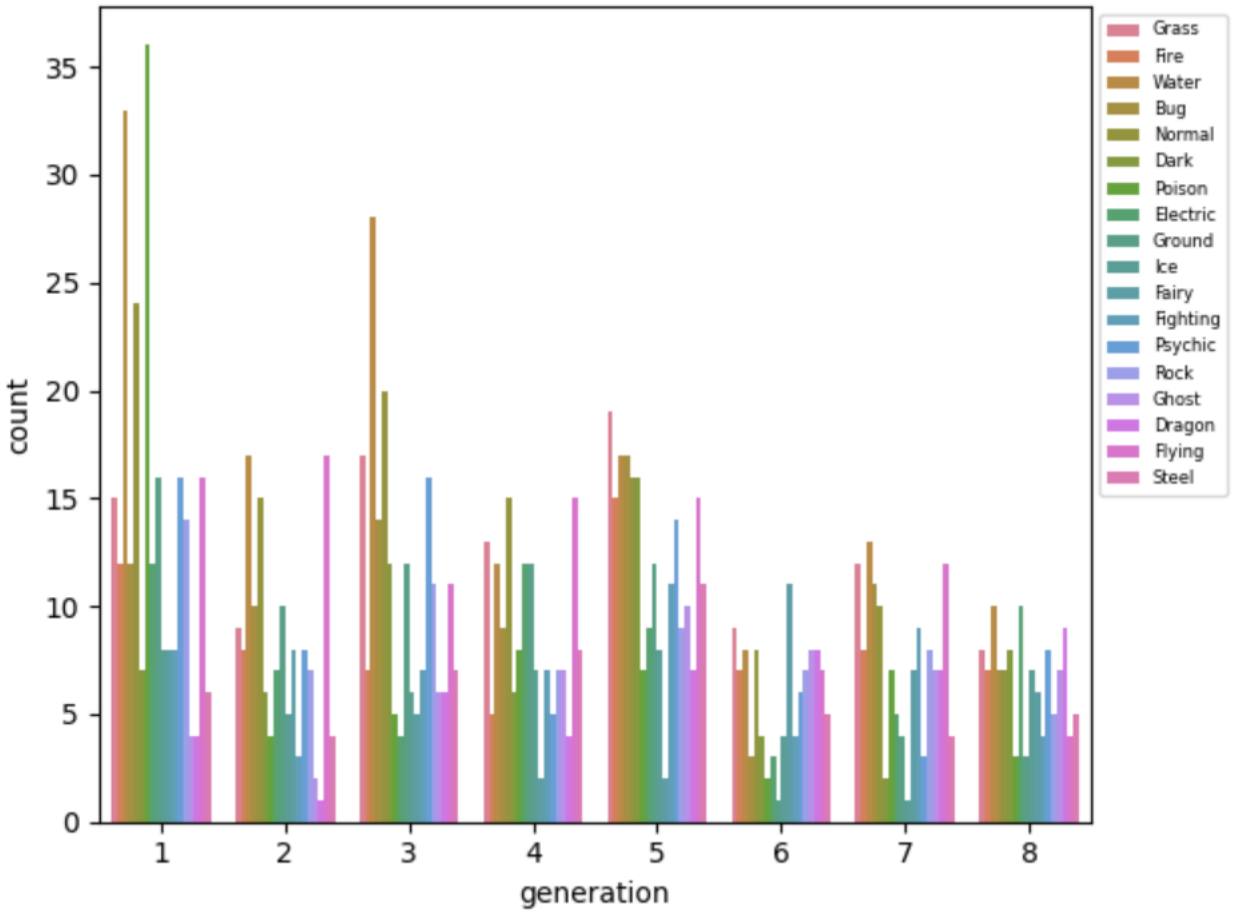


The second analysis is on types (type1 and type2) and its distribution variance through each generation, regardless of whether a Pokémon is single or double typed. The hypothesis is as follows:

H0: There is a near even to even distribution of Pokémon typing in each generation.

H1: There is no such correlation between generation and Pokémon types. They are not evenly distributed.

From our observations previously, majority of the main primary types, namely water, bug, and normal, are dominant in numbers throughout. In our exploratory analysis, we concluded that vast majority of the smaller amount in type1 are seen as dominant in type2 such as Flying. We will calculate the counts for both type1 and type2 and then combine them into a single series. Then, we will separate by type1 and type2 and concatenate the two DataFrame together. The resulting plot is as follows:



The count plot presented put into visual of the typing distribution throughout each generation. Keep in mind that each generation sees a decrease in new Pokémon. Interestingly, the majority of the Poison types seem to be introduced in generation 1. A contingency chart is then calculated:

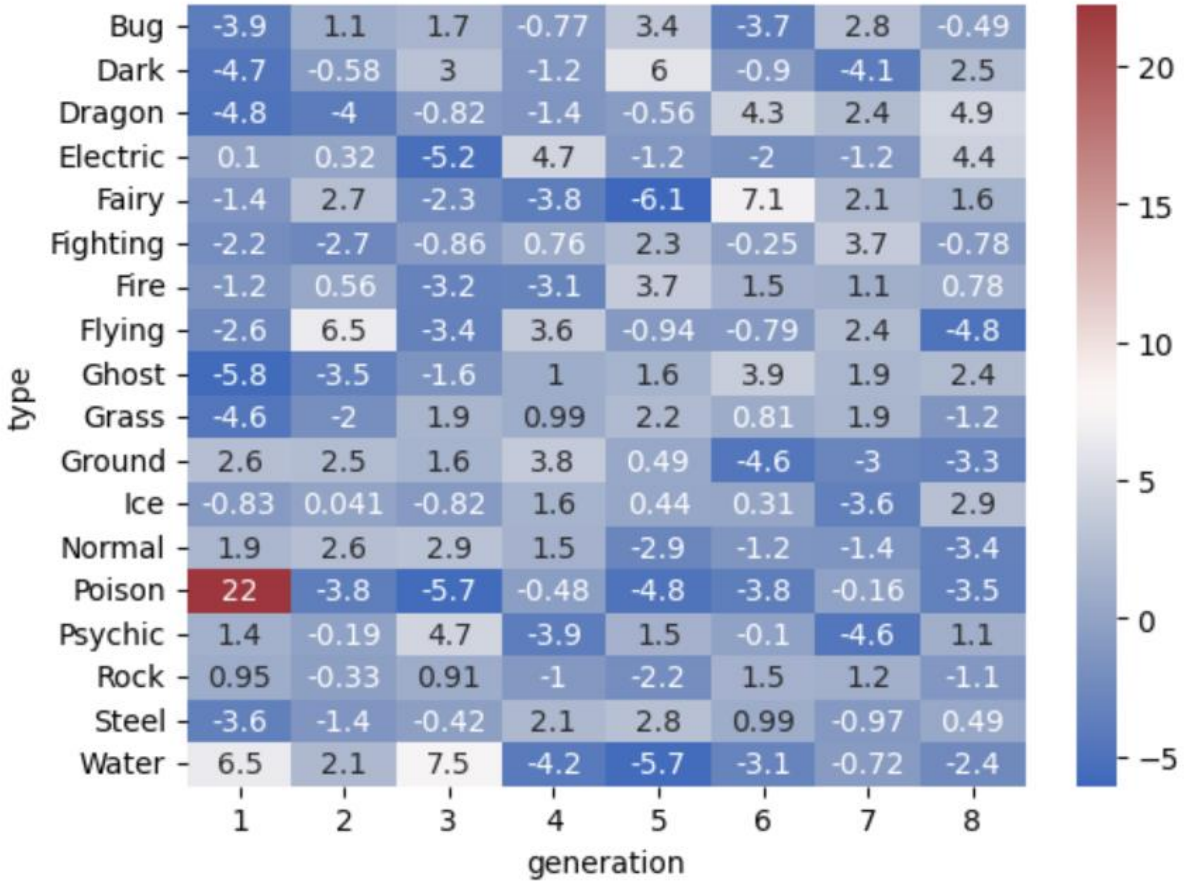
generation type	1	2	3	4	5	6	7	8
Bug	12	10	14	9	17	3	11	7
Dark	7	6	12	6	16	4	2	8
Dragon	4	1	6	4	7	8	7	9
Electric	12	7	4	12	9	3	5	10
Fairy	8	8	5	2	2	11	7	6
Fighting	8	3	7	7	11	4	9	4
Fire	12	8	7	5	15	7	8	7
Flying	16	17	11	15	15	7	12	4
Ghost	4	2	6	7	10	8	7	7
Grass	15	9	17	13	19	9	12	8
Ground	16	10	12	12	12	1	4	3
Ice	8	5	6	7	8	4	1	7
Normal	24	15	20	15	16	8	10	7
Poison	36	4	5	8	7	2	7	3
Psychic	16	8	16	5	14	6	3	8
Rock	14	7	11	7	9	7	8	5
Steel	6	4	7	8	11	5	4	5
Water	33	17	28	12	17	8	13	10

And then we calculate the chi2_contingency. Only the first few values will be provided again.

```
Chi2ContingencyResult(statistic=188.95018519409567, pvalue=4.705866615655822e-05, dof=119, e
xpected_freq=array([[15.92737003,  8.94724771, 12.31039755,  9.77217125, 13.64296636,
         6.66284404,  8.24923547,  7.48776758],
       [11.70565749,  6.57568807,  9.04740061,  7.18195719, 10.02675841,
         4.89678899,  6.06269113,  5.5030581 ],
       [ 8.82721713,  4.9587156 ,  6.82262997,  5.41590214,  7.56116208,
         3.69266055,  4.57186544,  4.14984709],
       [11.89755352,  6.68348624,  9.19571865,  7.29969419, 10.1911315 ,
         4.97706422,  6.16207951,  5.59327217],
       [ 9.4029052 ,  5.28211009,  7.2675841 ,  5.76911315,  8.05428135,
         3.93348624,  4.87003058,  4.4204893 ],
       [10.1704893 ,  5.71330275,  7.86085627,  6.24006116,  8.7117737 ,
         4.25458716,  5.2675841 ,  4.78134557],
       [13.24082569,  7.43807339, 10.23394495,  8.12385321, 11.34174312,
         5.53899083,  6.85779817,  6.22477064],
       [18.61391437, 10.45642202, 14.38685015, 11.4204893 , 15.9441896 ,
         7.78669725,  9.64067278,  8.75076453],
```

The p-value is a very small number again, which indicates that the null hypothesis is not true. There is no even distribution of typing between generations.

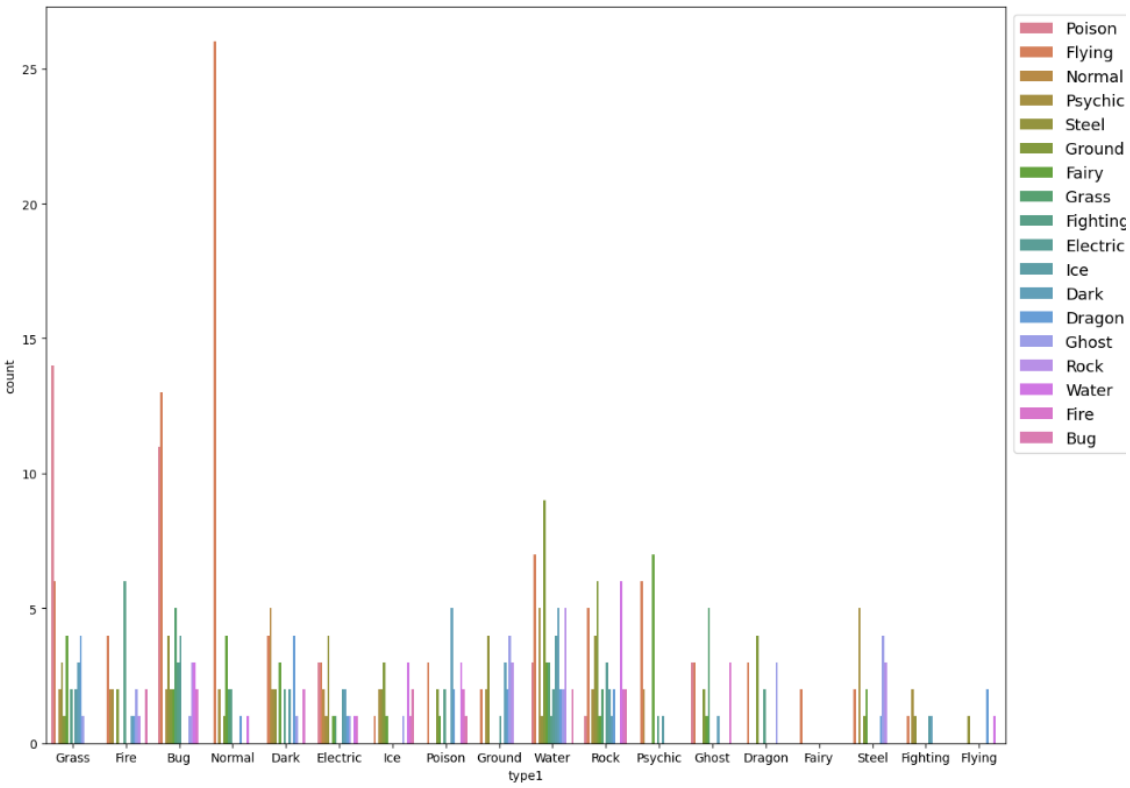
```
[278]:
<Axes: xlabel='generation', ylabel='type'>
```



The last chi-square that that will be calculated is type1 against type2. The following hypothesis is given:

H0: There is an even distribution of type2 with type1 Pokémon.

H1: There is an uneven distribution of type2 with type1 Pokémon.

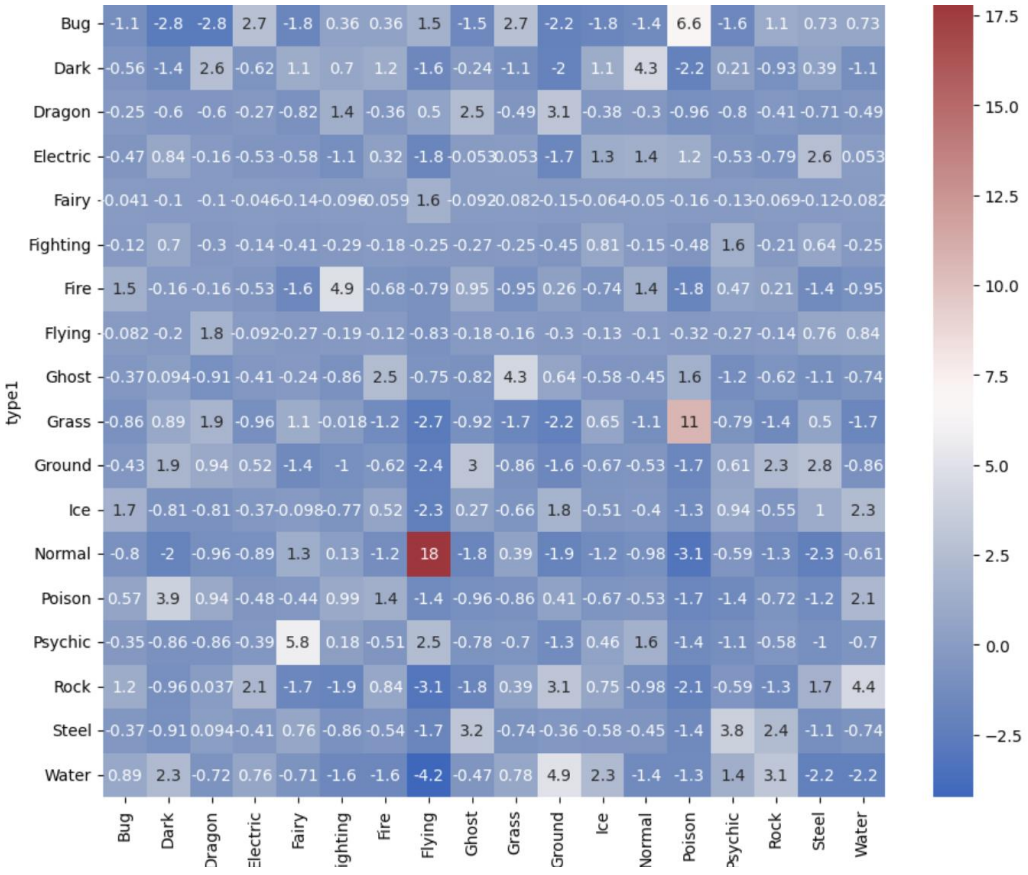


Upon observing this chart, it appears that most flying secondary types have normal as their primary type.

type2 / type1	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water
Bug	0	0	0	4	2	3	2	13	1	5	2	0	0	11	2	3	4	3
Dark	0	0	4	0	3	2	2	4	1	0	0	2	5	0	2	0	2	0
Dragon	0	0	0	0	0	2	0	3	3	0	4	0	0	0	0	0	0	0
Electric	0	2	1	0	1	0	1	3	1	1	0	2	2	3	1	0	4	1
Fairy	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
Fighting	0	1	0	0	0	0	0	1	0	0	0	1	0	0	2	0	1	0
Fire	2	1	1	0	0	6	0	4	2	0	2	0	2	0	2	1	0	0
Flying	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Ghost	0	1	0	0	1	0	3	3	0	5	2	0	0	3	0	0	0	0
Grass	0	3	4	0	4	2	0	6	1	0	1	2	0	14	2	0	3	0
Ground	0	3	2	1	0	0	0	2	4	0	0	0	0	0	2	3	4	0
Ice	2	0	0	0	1	0	1	1	1	0	3	0	0	0	2	0	2	3
Normal	0	0	1	0	4	2	0	26	0	2	1	0	0	0	2	0	0	1
Poison	1	5	2	0	1	2	2	3	0	0	2	0	0	0	0	0	0	3
Psychic	0	0	0	0	7	1	0	6	0	0	0	1	2	0	0	0	0	0
Rock	2	1	2	3	1	0	2	5	0	2	6	2	0	1	2	0	4	6
Steel	0	0	1	0	2	0	0	2	4	0	1	0	0	0	5	3	0	0
Water	2	5	2	2	3	1	0	7	2	3	9	4	0	3	5	5	1	0

```
[299]: Chi2ContingencyResult(statistic=631.6056350537023, pvalue=1.291485357834044e-27, dof=289, expected_freq=array([[ 1.13272311,  2.76887872,  2.76887872,
       1.25858124,  3.77574371,
               2.64302059,  1.63615561, 11.45308924,  2.51716247,  2.26544622,
               4.15331808,  1.76201373,  1.38443936,  4.40503432,  3.64988558,
               1.88787185,  3.27231121,  2.26544622],
              [ 0.55606407,  1.35926773,  1.35926773,  0.61784897,  1.85354691,
               1.29748284,  0.80320366,  5.62242563,  1.23569794,  1.11212815,
               2.0389016 ,  0.86498856,  0.67963387,  2.1624714 ,  1.79176201,
               0.92677346,  1.60640732,  1.11212815],
              [ 0.24713959,  0.60411899,  0.60411899,  0.27459954,  0.82379863,
               0.57665904,  0.35697941,  2.49885584,  0.54919908,  0.49427918,
               0.90617849,  0.38443936,  0.3020595 ,  0.9610984 ,  0.79633867,
               0.41189931,  0.71395881,  0.49427918],
              [ 0.47368421,  1.15789474,  1.15789474,  0.52631579,  1.57894737,
               1.10526316,  0.68421053,  4.78947368,  1.05263158,  0.94736842,
               1.73684211,  0.73684211,  0.57894737,  1.84210526,  1.52631579,
               0.78947368,  1.36842105,  0.94736842],
              [ 0.04118993,  0.1006865 ,  0.1006865 ,  0.04576659,  0.13729977,
               0.09610984,  0.05949657,  0.41647597,  0.09153318,  0.08237986,
               0.15102975,  0.06407323,  0.05034325,  0.16018307,  0.13272311,
               0.06864989,  0.11899314,  0.08237986],
              [ 0.12356979,  0.3020595 ,  0.3020595 ,  0.13729977,  0.41189931,
               0.28832952,  0.1784897 ,  1.24942792,  0.27459954,  0.24713959,
               0.45308924,  0.19221968,  0.15102975,  0.4805492 ,  0.39816934,
               0.20594966,  0.35697941,  0.24713959],
```

The chi2_contingency calculation also shows low p-value which means our null hypothesis can again be rejected.



To conclude this analysis, I have three points I need to make:

1) Dual typing is not random. There seems to be type compatibility and only certain types work with another.

2) There is uneven distribution of certain types, and they appear more frequently than others.
3) Generation and regional geography influence which kind of Pokémon inhabits it. Due to this factor, the chance that these Pokémon are randomly generated/created is slim. Great attention to detail is placed into making these fictional biological creatures.

3. K-means Clustering

I am interested to see how the values are presented and separated in the cluster analysis. The Pokedex numbers Pokémon as they are created. The earlier generations fit in the lower ID ranges, and the later ones in the higher number ranges. Is there a significant difference in values per cluster?

I first started off with 2 clusters. Originally the data was preprocessed with one-hot encoding for the classifications, but it results in over 600 rows of data that I do not believe I will be able to use efficiently. I also removed the pokedex number and generation as that may mess up the algorithm. I only incorporated the types and decided to do a type analysis with stats distribution.

cluster	0	1
height_m	0.572238	1.449035
weight_kg	14.731728	76.70444
base_total	304.764873	479.355212
hp	50.124646	77.787645
attack	54.376771	88.171815
defense	51.730878	82.22973
sp_attack	48.393768	78.832046
sp_defense	49.509915	79.374517
speed	50.628895	72.959459
capture_rate	172.685552	59.453668
type1_Bug	0.096317	0.07722
type1_Dark	0.042493	0.046332
type1_Dragon	0.01983	0.032819
type1_Electric	0.065156	0.055985
type1_Fairy	0.025496	0.019305
type1_Fighting	0.031161	0.040541
type1_Fire	0.050992	0.073359
type1_Flying	0.008499	0.005792
type1_Ghost	0.031161	0.03861
type1_Grass	0.104816	0.090734
type1_Ground	0.048159	0.03861
type1_Ice	0.033994	0.03861
type1_Normal	0.141643	0.104247
type1_Poison	0.048159	0.03861
type1_Psychic	0.062323	0.048263
type1_Rock	0.03966	0.07529
type1_Steel	0.016997	0.034749
type1_Water	0.133144	0.140927
type2_Bug	0.014164	0.007722
type2_Dark	0.022663	0.027027
type2_Dragon	0.011331	0.034749
type2_Electric	0.011331	0.011583
type2_Fairy	0.036827	0.032819
type2_Fighting	0.008499	0.034749
type2_Fire	0.011331	0.017375
type2_Flying	0.084986	0.117761
type2_Ghost	0.016997	0.027027
type2_Grass	0.025496	0.017375
type2_Ground	0.028329	0.044402
type2_Ice	0.002833	0.025097
type2_Normal	0.011331	0.013514
type2_Poison	0.048159	0.034749
type2_Psychic	0.025496	0.03861
type2_Rock	0.011331	0.021236
type2_Steel	0.016997	0.03861
type2_Water	0.022663	0.019305

Looking at this data, the separation of values is 353 to 518 through a simple count query. The first cluster's Pokémon has the lowest stats, while the second has higher stats and physiques. Besides the capture rate, all of the other stats went up, which went down. This is normal because the stronger a Pokémon is, the less likely it can be caught with a normal Pokéball. It seems like the types that align the most with the stat change analysis would be

Dragon, Fighting, Rock, and Steel. This is based off both types increases in concentration. There was notable increase in concentration for both Type 1 and Type 2.

For my second clustering test, I decided to plot an inertia graph to help me figure out how many clusters I should experiment with. The elbow of the graph suggests 4 to 9, so I will start with 5. After a couple of recycling, 233:236:45:352:5 respective to clusters 0 to 4 would be the best ratio for counts that I can do. I believe the large difference is because there are not a lot of Pokémon that have near maximal stats.

clusters	0	1	2	3	4
height_m	0.787983	0.537288	3.064444	1.359659	5.14
weight_kg	23.999571	13.261017	281.262222	54.764205	855.58
base_total	376.785408	285.398305	511.844444	497.159091	542.0
hp	61.180258	47.826271	91.022222	78.826705	120.6
attack	68.93133	49.665254	104.044444	90.386364	117.6
defense	63.167382	49.716102	107.933333	82.420455	107.2
sp_attack	61.11588	44.351695	73.911111	83.786932	78.4
sp_defense	61.236052	46.694915	80.0	83.227273	81.8
speed	61.154506	47.144068	54.933333	78.511364	36.4
capture_rate	82.081545	210.211864	53.577778	57.78125	54.0
type1_Bug	0.085837	0.122881	0.066667	0.0625	0.0
type1_Dark	0.055794	0.042373	0.0	0.042614	0.2
type1_Dragon	0.051502	0.0	0.022222	0.03125	0.0
type1_Electric	0.060086	0.050847	0.044444	0.068182	0.0
type1_Fairy	0.008584	0.033898	0.0	0.025568	0.0
type1_Fighting	0.017167	0.042373	0.044444	0.045455	0.0
type1_Fire	0.098712	0.029661	0.044444	0.068182	0.0
type1_Flying	0.004292	0.008475	0.0	0.008523	0.0
type1_Ghost	0.038627	0.021186	0.044444	0.042614	0.0
type1_Grass	0.11588	0.088983	0.044444	0.096591	0.0
type1_Ground	0.04721	0.042373	0.066667	0.034091	0.2
type1_Ice	0.025751	0.033898	0.088889	0.039773	0.0
type1_Normal	0.128755	0.152542	0.044444	0.102273	0.0
type1_Poison	0.021459	0.055085	0.0	0.053977	0.0
type1_Psychic	0.030043	0.076271	0.0	0.0625	0.0
type1_Rock	0.06867	0.029661	0.222222	0.053977	0.2
type1_Steel	0.017167	0.021186	0.111111	0.022727	0.4
type1_Water	0.124464	0.148305	0.155556	0.139205	0.0
type2_Bug	0.008584	0.012712	0.0	0.011364	0.0
type2_Dark	0.017167	0.016949	0.044444	0.034091	0.0
type2_Dragon	0.017167	0.008475	0.133333	0.025568	0.2
type2_Electric	0.008584	0.012712	0.022222	0.011364	0.0
type2_Fairy	0.025751	0.04661	0.0	0.036932	0.0
type2_Fighting	0.021459	0.008475	0.022222	0.036932	0.0
type2_Fire	0.021459	0.004237	0.022222	0.017045	0.0
type2_Flying	0.107296	0.084746	0.066667	0.119318	0.2
type2_Ghost	0.030043	0.016949	0.022222	0.022727	0.0
type2_Grass	0.025751	0.016949	0.022222	0.019886	0.0
type2_Ground	0.042918	0.025424	0.155556	0.028409	0.0
type2_Ice	0.012876	0.0	0.088889	0.019886	0.0
type2_Normal	0.008584	0.016949	0.0	0.014205	0.0
type2_Poison	0.04721	0.050847	0.022222	0.03125	0.0
type2_Psychic	0.021459	0.021186	0.066667	0.045455	0.0
type2_Rock	0.017167	0.008475	0.066667	0.017045	0.0
type2_Steel	0.017167	0.016949	0.044444	0.042614	0.2
type2_Water	0.021459	0.021186	0.0	0.022727	0.0
cluster	0.497854	0.0	1.0	1.0	1.0

Immediately, you can see the existence of 3/4 types we inferred that have really high stats, being Rock, Steel, and Dragon. Ground, Dark, and Flying made it as well. The separation of Type 1 and Type 2 suggests that some of these strong Pokémon may be dual typed. Furthermore, we have now a clearer data distribution. The types that seem more prominent in the near average clusters 2 and 3 are notably Electric, Grass, Ground, Ice, and Normal for type 1, and Dragon, Flying, Ground, Ice, and Psychic for type 2.

I did a short research and found that someone else compiled a set of data 4 years ago, about half a year after the release date of Pokémon Sword and Shield (8th generation), that displays the average stats of each type on Reddit (https://www.reddit.com/r/TruePokemon/comments/gizsff/average_base_stat_comparison_by_type_posted_this/). Assuming they ran the data on all of the Pokémon from generation 1 to 8, the following observation is as follows for top 5 highest average stat in order:

Hp – Dragon, Ice, Normal, Ground, Fighting

Attack – Fighting, Dragon, Dark, Steel, Ground

Defense – Steel, Rock, Dragon, Ground, Ghost

Special Attack – Dragon, Psychic, Fire, Electric, Fairy

Special Defense – Psychic, Fairy, Dragon, Steel, Ghost

Speed – Flying, Electric, Dragon, Psychic, Fighting

Stat Total – Dragon, Steel, Psychic, Fighting, Ice

Dragon types appear in all the highest average stats ranking. We can see this in our clusters, where the type was mentioned several times and had a high concentration percentage in the clusters higher-up. The next common ones are Steel, Fighting, Ground, and Psychic. These four types also popped up on our observations. It is likely these 5 types can be stand-alone typing (have only 1 type), or they can be combined in a dual type. The types that fill only 1 of the top 5 stat rankings most likely will need to be dual types that came with one of the types with higher statistics, if we take into consideration our findings.

4 & 5. Linear Regression between two variables, PearsonR

I am curious on how there are relationship between two variables and if it's effective in predicting each other. I decided to use linear regress on the following variables: Attack vs Defense, Sp. Attack vs. Sp. Defense, Capture Rate vs Base Total, Weight vs Speed, Sp. Attack vs. Attack, Hp vs. Defense. Since some of these comparisons (like sp. Attack vs sp. Defense and attack vs defense), I will only focus on 1 of them in the last two comparisons.

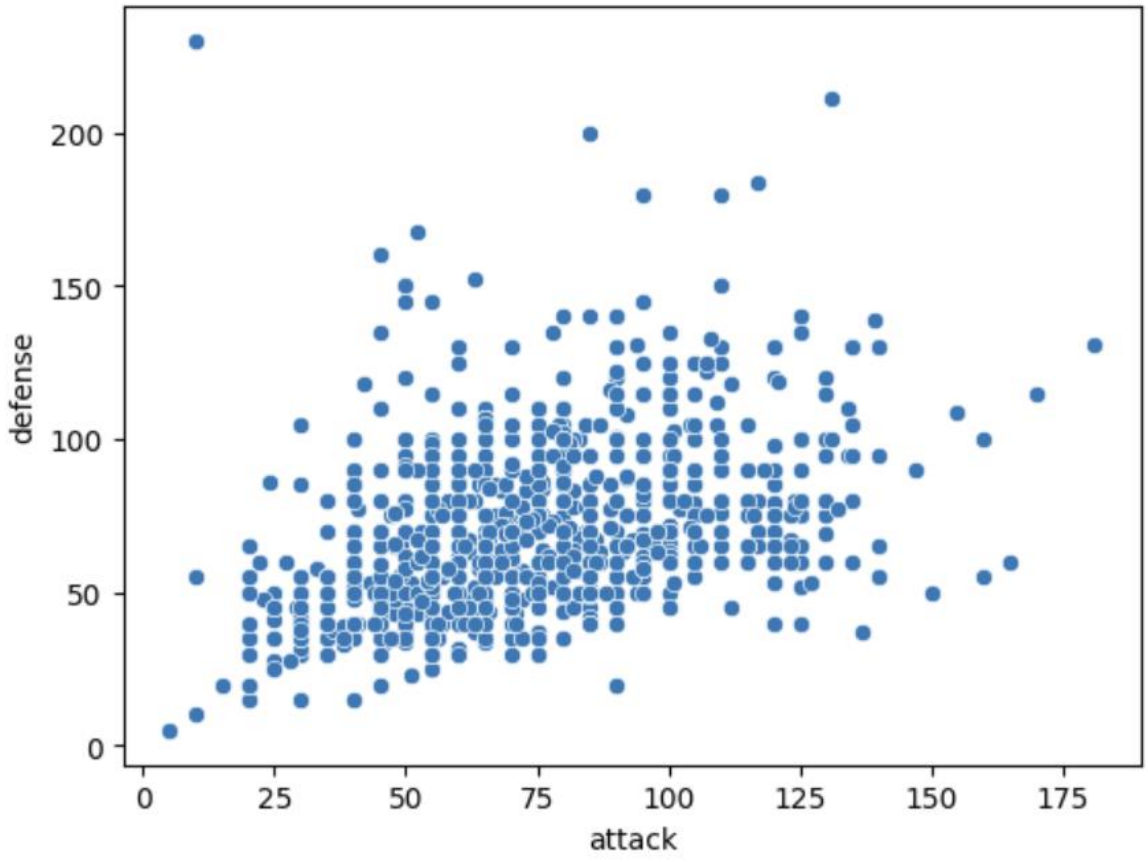
Before we process anything, the numeric variables have to be converted to float because Seaborn's regplot detected inconsistency of datatypes in the data. I made a copy of the pkmn_df dataframe to accommodate those changes and not affect everything else that has been done so far.

Attack vs. Defense

Scatterplot:

```
sns.scatterplot(pkmn_floatconv, x='attack', y='defense')
```

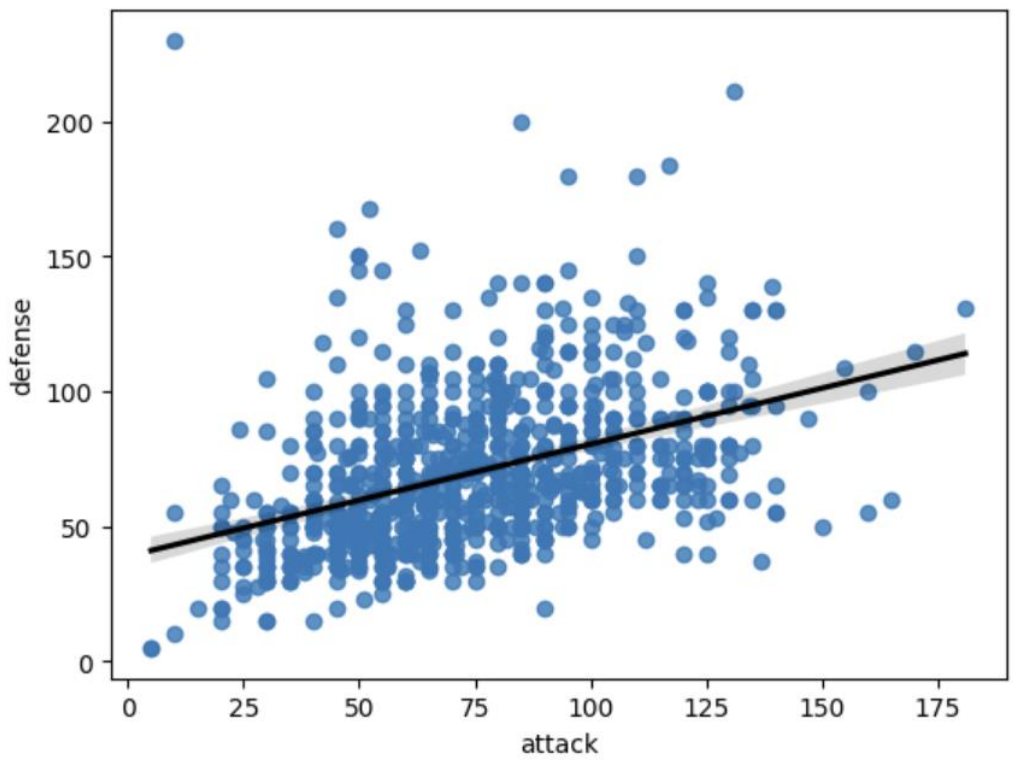
```
<Axes: xlabel='attack', ylabel='defense'>
```



Regplot

```
sns.regplot(pkmn_floatconv, x='attack', y='defense', line_kws={'color': 'black'})
```

```
<Axes: xlabel='attack', ylabel='defense'>
```



There is a linear regression line. It seems like there is some sort of correlation due to the slope. Let's calculate the linear regression.

```
linreg_results = stats.linregress(pkmn_floatconv['attack'], pkmn_floatconv['defense'])
linreg_results
```

```
LinregressResult(slope=0.425328637668305, intercept=44.7579798232579, rvalue=0.41989208696821556, pvalue=1.608356242235378e-38, stderr=0.031185959515663,
intercept stderr=2.3537251252405844)
```

We are getting both the linear regression results and Pearsonr result, if you look at the rvalue and pvalue. I ran a query to see if they are the same and indeed they are.

```
stats.pearsonr(pkmn_floatconv['attack'], pkmn_floatconv['defense'])
```

```
PearsonRResult(statistic=0.4198920869682154, pvalue=1.6083562422355335e-38)
```

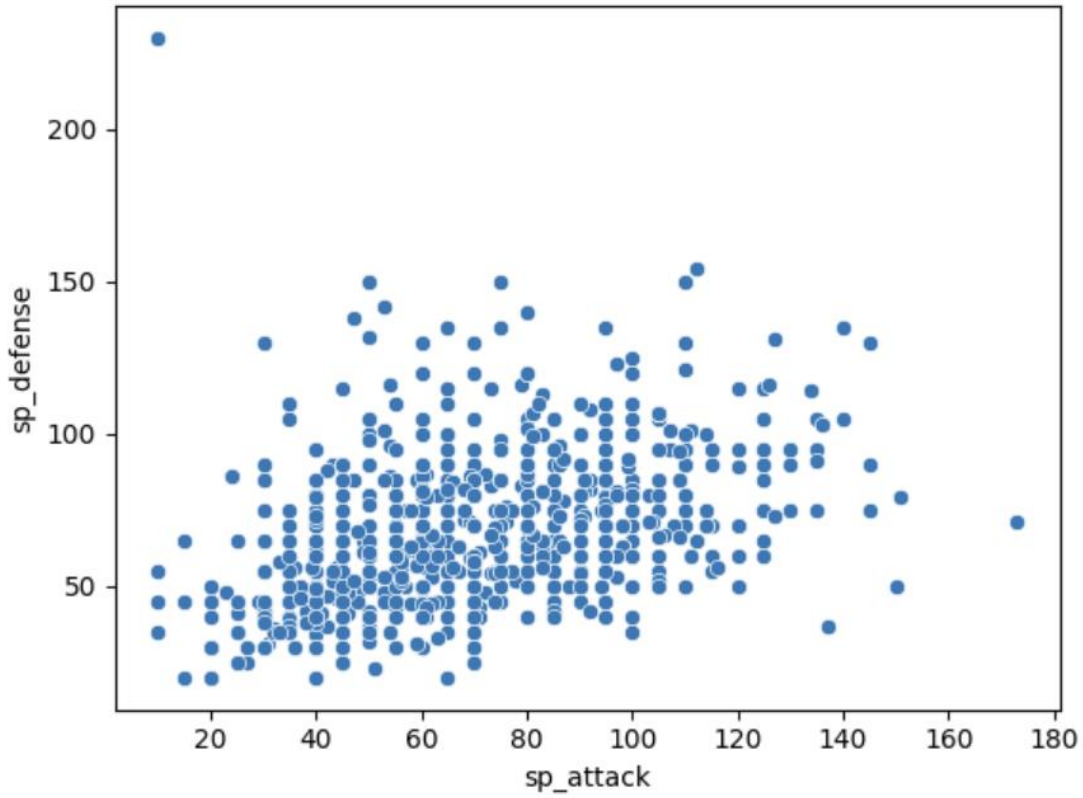
The statistic/r value is between 0.3 and 0.7 at 0.42, which means that the two variables have a moderate linear relationship. Indeed, as attack goes up, defense also goes up and vice versa. This can also be interpreted by the slop of 0.425, in which it is not near y = 0 horizontal or x = x vertical. The p-value is significantly low, which means we can reject the null hypothesis of non-correlation.

Sp .Attack vs. Sp. Defense

Sp.Attack and Sp. Defense correlation yields very similar results to attack vs. defense.

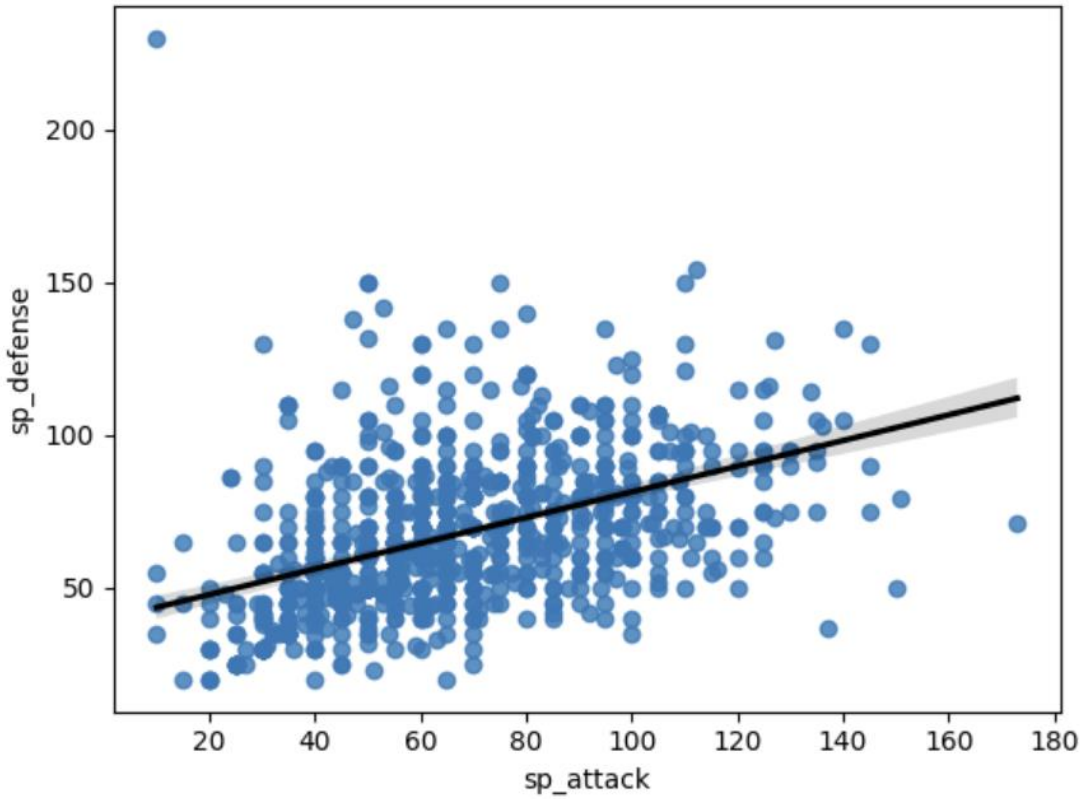
```
pkmn_floatconv['sp_attack'] = pkmn_floatconv['sp_attack'].astype(float)
pkmn_floatconv['sp_defense'] = pkmn_floatconv['sp_defense'].astype(float)
sns.scatterplot(pkmn_floatconv, x='sp_attack', y='sp_defense')
```

```
<Axes: xlabel='sp_attack', ylabel='sp_defense'>
```



```
sns.regplot(pkmn_floatconv, x='sp_attack', y='sp_defense', line_kws={'color': 'black'})
```

```
<Axes: xlabel='sp_attack', ylabel='sp_defense'>
```




```
linreg_results = stats.linregress(pkmn_floatconv['attack'], pkmn_floatconv['defense'])
linreg_results
```

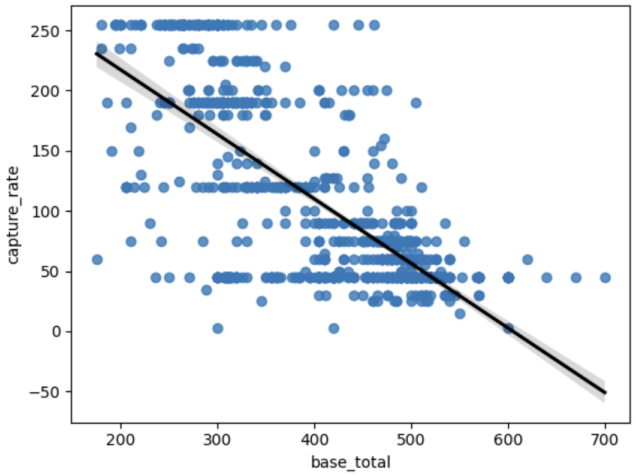
```
LinregressResult(slope=0.41452502625985743, intercept=38.997233750373795, rvalue=0.41989208696821556, pvalue=1.608356242235378e-38, stderr=0.030393816786
093143, intercept_stderr=2.4281626915291894)
```

Not much can be said here, except what was said for attack vs. defense. This would suggest that any attack/sp.attack versus their defense counterparts are positively correlated.

Base Total vs. Capture Rate

```
sns.regplot(pkmn_floatconv, x='base_total', y='capture_rate', line_kws={'color': 'black'})
```

```
<Axes: xlabel='base_total', ylabel='capture_rate'>
```




```
linreg_results = stats.linregress(pkmn_floatconv['base_total'], pkmn_floatconv['capture_rate'])
linreg_results
```

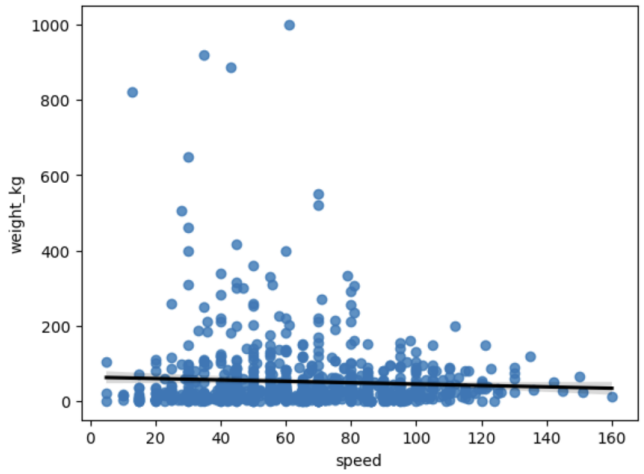
```
LinregressResult(slope=-0.5359674146697414, intercept=324.33911741904126, rvalue=-0.7193215204015634, pvalue=1.2146148640508441e-139, stderr=0.0175585460
95839307, intercept_stderr=7.384541846843259)
```

The graph was plotted with a visible linear regression line through the scatter plot. The slope is negative (-0.54), which means there is an effect of the base total on the capture rate, except the higher the base total, the lower the capture rate. This is especially true because the strong the Pokémon, the harder it is to be caught by a regular pokeball. The Pearsonr value suggests that this is a strong, negative relationship and the p-value is significantly small, which means the null hypothesis of non-correlation is rejected.

Speed vs. Weight

```
pkmn_floatconv['weight_kg'] = pkmn_floatconv['weight_kg'].astype(float)
pkmn_floatconv['speed'] = pkmn_floatconv['speed'].astype(float)
sns.regplot(pkmn_floatconv, x='speed', y='weight_kg', line_kws={'color': 'black'})
```

```
<Axes: xlabel='speed', ylabel='weight_kg'>
```




```
linreg_results = stats.linregress(pkmn_floatconv['speed'], pkmn_floatconv['weight_kg'])
linreg_results
```

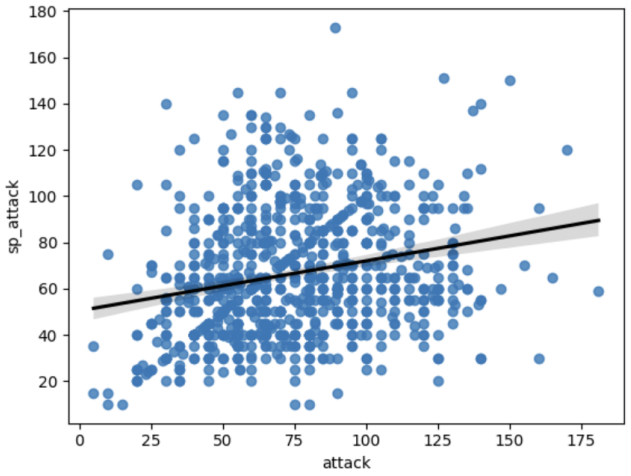
```
LinregressResult(slope=-0.18444169838357027, intercept=63.37559947247007, rvalue=-0.05458742998893435, pvalue=0.10741754030749416, stderr=0.1144480922888
9949, intercept_stderr=7.950336855464071)
```

At first glance, we see that the line is going downward, which means there's a negative slope so the higher the speed, the less the Pokémon weighs. But then we look at the Pearsonr result. The r value is significantly low, below -0.3, which means that the relationship is very weak. The pvalue is over 0.05, which means that we fail to reject the null hypothesis of non-correlation, but there is also not enough evidence to support the alternative hypothesis where states that these two variables are correlated. As in a previous prediction using the correlation matrices, we also came to this conclusion, but now linear regression and Pearsonr has given us more insight.

Attack vs Sp. Attack

```
sns.regplot(pkmn_floatconv, x='attack', y='sp_attack', line_kws={'color': 'black'})
```

```
<Axes: xlabel='attack', ylabel='sp_attack'>
```




```
linreg_results = stats.linregress(pkmn_floatconv['attack'], pkmn_floatconv['sp_attack'])
linreg_results
```

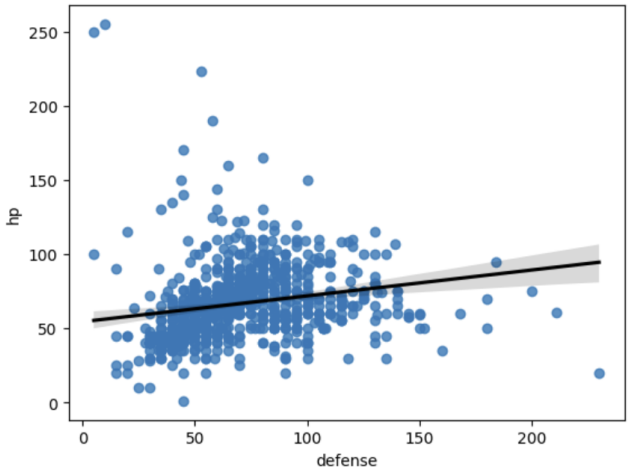
```
LinregressResult(slope=0.21581252176099483, intercept=50.42327593387805, rvalue=0.22757383884956903, pvalue=1.0753190174932266e-11, stderr=0.031325406484
79879, intercept_stderr=2.50258741304837)
```

Upon observation, attack vs sp. Attack has weak positive correlation due to low rvalue and small slope. However, we can reject the null hypothesis because the p-value is small. In the Pokémon world, Attack and Special attack points increase as a Pokémon grows stronger, but it does not usually mean that a Pokémon will have a similar amount of attack and sp. Attack points. For example, a normal type Pokémon stats would have more physical attack then special attack because it's not considered "magical". The same can be said for defense and sp. Defense.

Hp vs. Defense

```
pkmn_floatconv['hp'] = pkmn_floatconv['hp'].astype(float)
sns.regplot(pkmn_floatconv, x='hp', y='defense', line_kws={'color': 'black'})
```

```
<Axes: xlabel='defense', ylabel='hp'>
```




```
linreg_results = stats.linregress(pkmn_floatconv['hp'], pkmn_floatconv['defense'])
linreg_results
```

```
LinregressResult(slope=0.2299460832722959, intercept=54.560145262004724, rvalue=0.20017554116030373, pvalue=2.5276526615432654e-09, stderr=0.038179025881
21611, intercept_stderr=2.7130610876057015)
```

Like, attack vs. sp. Attack, the correlation is weak between hp and defense. A pokemon can have higher defense if they have higher hp usually, but the weak correlation tells us it’s not always the case.

6. K-nearest neighbor

How can I use the current data to predict a Pokémon type if given the height, weight, hp, attack, defense, special attack, special defense, and speed? I tried to use k-nearest neighbor to help test the accuracy of the data set in predictions. Initially, I tried to use multi-class classification. The Train test split was: training size 341, validation size 52, test size 44. I dropped all of the columns except the ones mentioned above as they are non-significant column variables. I also normalized the numeric columns due to an inconsistent scale between base total and the rest.

```
[716]: pkmn_mod = pkmn_df.dropna().reset_index()
pkmn_mod = pkmn_mod.drop(['level_0','index','pokedex_number','name','classfication','generation','type2','capture_rate'], axis=1)


#normalized
num_col = ['height_m', 'weight_kg', 'base_total', 'hp', 'attack', 'defense', 'sp_attack', 'sp_defense', 'speed']
pkmn_type = pkmn_mod['type1']
pkmn_mod_num = pd.DataFrame(pkmn_mod[num_col])
pkmn_mod_norm = pd.concat([
    (pkmn_mod_num - pkmn_mod_num.mean())/pkmn_mod_num.std(),
    pkmn_type
], axis=1)
pkmn_mod_norm
```

[716]:

	height_m	weight_kg	base_total	hp	attack	defense	sp_attack	sp_defense	speed	type1
0	-0.437973	-0.531576	-1.095254	-0.993306	-0.959093	-0.835172	-0.16859	-0.199144	-0.759283	Grass
1	-0.170787	-0.472338	-0.206791	-0.340693	-0.512376	-0.386333	0.371012	0.385708	-0.208109	Grass
2	0.719832	0.372545	1.018674	0.529458	0.174881	0.254865	1.090482	1.165512	0.526789	Grass
3	0.452647	0.280287	1.110584	0.442443	0.243607	0.094566	1.414243	0.580659	1.261688	Fire
4	-0.081725	-0.287823	-0.308913	-0.340693	-1.096544	-0.803112	0.730747	0.385708	0.15934	Bug
5	-0.79422	-0.567508	-2.351355	-1.210844	-1.440173	-1.444311	-1.787396	-1.953702	-0.575558	Bug
6	-0.527035	-0.501471	-2.249233	-0.993306	-1.783801	-0.803112	-1.607529	-1.758751	-1.126732	Bug
7	-0.170787	-0.312101	-0.308913	-0.123155	0.449784	-1.123712	-0.888059	0.385708	0.343065	Bug
8	-0.79422	-0.581104	-1.779472	-1.210844	-1.096544	-1.123712	-1.247794	-1.36885	-0.355089	Normal

Then I ran the k-nearest neighbors training classifiers, and used it against the validation set to computer the accuracy. This was the result:

[737]:
```
for K in range(1, 10):
  #train k neighbors classifier and train
  knn = KNeighborsClassifier(n_neighbors=K)
  knn.fit(X_train, y_train)

  #compute accuracy
  predictions = knn.predict(X_val)
  print('k:', K, 'accuracy:', metrics.accuracy_score(y_val, predictions))
```

```
k: 1 accuracy: 0.19230769230769232
k: 2 accuracy: 0.1346153846153846
k: 3 accuracy: 0.19230769230769232
k: 4 accuracy: 0.21153846153846154
k: 5 accuracy: 0.1346153846153846
k: 6 accuracy: 0.15384615384615385
k: 7 accuracy: 0.17307692307692307
k: 8 accuracy: 0.19230769230769232
k: 9 accuracy: 0.17307692307692307
```

Then I ran it on the rest, with n_neighbors=4 that had the highest computed accuracy of ~0.27.

[738]:
```
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train, y_train)
metrics.accuracy_score(y_test, knn.predict(X_test))
#despite the accuracy, 4th has the better result of ~0.27
```

[738]:
```
0.2727272727272727
```

I ran a test, 2-row dataframe.

[802]:
```
#Let's run it on a sample Pokemon, with stats randomly generated
test_pkmn = pd.DataFrame({'height_m': [0.4, 1.4], 'weight_kg': [1.0, 50], 'base_total': [203, 362], 'hp': [52, 90],
                          'attack': [36, 100] , 'defense': [55, 70] , 'sp_attack': [40, 57], 'sp_defense': [20,45], 'speed': [30, 50]}, [1,2])
```

[803]:
```
#Is it a Bug type? Is the second Pokemon a Fighting type?
metrics.accuracy_score(pd.Series(['Bug', 'Fighting']), knn.predict(test_pkmn))
```

[803]:
```
0.0
```

Because the accuracy is so small, it is difficult to figure out whether this can be useful in future predictions. However, this model does have at most 27% chance in succeeding for classification.

I tried to use the Principle Component Analysis. However, it did not really help much and the predictions were similar.

Later, I attempted a binary classification. Is the presented Pokémon a water type or no? It was done on water types specifically because type 1 had the most amount of Pokémon that are water type.

First, I had to convert all of the types to “Water” or “Not Water” and then I computed the normalization, and then the k-neighbors with the same set size.

```
for K in range(1, 8):
  #train k neighbors classifier and train
  knn = KNeighborsClassifier(n_neighbors=K)
  knn.fit(X_train_w, y_train_w)

  #compute accuracy
  predictions = knn.predict(X_val_w)
  print('k:', K, 'accuracy:', metrics.accuracy_score(y_val_w, predictions))
```

```
k: 1 accuracy: 0.8653846153846154
k: 2 accuracy: 0.9038461538461539
k: 3 accuracy: 0.8461538461538461
k: 4 accuracy: 0.8846153846153846
k: 5 accuracy: 0.8846153846153846
k: 6 accuracy: 0.9230769230769231
k: 7 accuracy: 0.9230769230769231
```

I fitted every k-neighbor that was produced and concluded that the 3rd and 5th n-neighbor seem to yield the higher accuracy of ~0.795.

```
[783]: knn = KNeighborsClassifier(n_neighbors=3)
       knn.fit(X_train_w, y_train_w)
       metrics.accuracy_score(y_test_w, knn.predict(X_test))
       #3rd and 5th seem to be best
```

```
[783]: 0.7954545454545454
```

I reused the test_pkmn dataframe I made of the two random Pokémon and the result was a 0.5 accuracy, in which 1 of the 2 was classified correctly.

```
[804]: metrics.accuracy_score(pd.Series(['Water', 'Not Water']), knn.predict(test_pkmn))
```

```
[804]: 0.5
```

Since the classifying accuracy is higher in this binary observation, this model is much more usable.

CONCLUSION

In conclusion, I learned the following key points:

- It is highly unlikely Pokémon are randomly generated. Each Pokémon has specific typing and if dual types, they have a specific match-up rather than random matching, different classifications that suit the Pokémon's overall physiques, and stats that coincide with what their typing suggests.
- There is suggested correlation relationship between all stats, but some are more prominent than others, such as attack vs defense compared to hp vs defense.
- Steel, Fighting, Ground, Psychic, and Dragon types have higher values of battle stats. Specifically, dragon types, although that was part of my prediction before this entire data analysis. Another key to note is that Dragon types are only weak to Dragon types, which makes them harder to take down.
- Pokémon classification and types are not evenly distributed between generations as shown in the chi square test. This is only because the Pokémon in this dataset have their generation recorded as first appearances. Specific Pokémon show up in later series. Since this is the case, we are not able to see the full picture. The dataset can be expanded on.
- Considering everything, this model can be used for prediction if we are just predicting 1-2 types. However, using multi-classes to predict is generally difficult and results in low accuracy due to noise and all the possible combinations. If possible, I would like to learn how to do it more efficiently.

I learned quite a lot from doing this project and a lot of what I noticed playing the game showed up in this analysis, along with a lot that I did not know. For example, when I played generation 5, I noticed that Scolipede is heavy in the game, yet it moves faster than all my other Pokémon. I was able to apply this knowledge to when I did the weight vs speed correlation test and prove that the correlation is very weak, and the null hypothesis of non-correlation has failed to be rejected. Something that I somewhat knew but gained more insight into is the similarity in stats of Pokémon with the same primary type. As seen during the k-nearest neighbor training algorithm, the accuracy was high because Pokémon with the same single typing tend to have similar battle stats, although their physical traits can be very different. If we added in the secondary type, we may also need to consider that for the prediction model which can make the training algorithm difficult. Typing influences a Pokémon's physiques and battle statistics and perhaps also their Pokémon species classification. I also believe that the data could have more information such as all the generations the Pokémon appeared in and maximum stat points. I think it may balance out the skewness of the data. Overall, this was a decent dataset to work with and it was enjoyable learning as well.

CITATIONS

- *Where legends come to life*. Serebii.net Header. (n.d.). https://www.serebii.net/

- *Main page*. Bulbapedia, the community-driven Pokémon encyclopedia. (n.d.). https://bulbapedia.bulbagarden.net/wiki/Main_Page

- *Pokémon database*. Pokémon Database -- the fastest way to get your Pokémon information. (n.d.). https://pokemondb.net/

- Average Base Stat Comparisons https://www.reddit.com/r/TruePokemon/comments/gizsff/average_base_stat_comparison_by_type_posted_this/